

z/OS  
3.2

*UNIX System Services  
Programming: Assembler Callable  
Services Reference*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 1453](#).

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 1996, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures.....</b>	<b>xxiii</b>
<b>Tables.....</b>	<b>xxv</b>
<b>About this document.....</b>	<b>xxix</b>
Who should use this document.....	xxix
z/OS information.....	xxix
Discussion list.....	xxix
<b>How to provide feedback to IBM.....</b>	<b>xxxi</b>
<b>Summary of changes.....</b>	<b>xxxiii</b>
Summary of changes for z/OS 3.2.....	xxxiii
Summary of changes for z/OS 3.1.....	xxxiii
<b>Chapter 1. Invocation details for callable services.....</b>	<b>1</b>
Connecting to and disconnecting from z/OS UNIX.....	1
Syntax conventions for the callable services.....	1
CALL.....	1
Service_name.....	2
Parm parameters.....	2
Return_value.....	2
Return_code.....	2
Reason_code.....	3
Determining the callable service release level.....	3
Linkage conventions for the callable services.....	3
Parameter descriptions for the callable services.....	4
Call parameter lists.....	4
Mapping macros.....	4
Examples.....	5
Reentrant coding versus nonreentrant coding.....	5
Environmental restrictions.....	5
Restrictions in a multiprocess, multiuser environment.....	6
Restrictions in a multiprocess environment.....	6
Abend conditions and environments.....	7
Callable service failures.....	7
Authorization.....	8
Using callable services in a 64-bit environment.....	8
Call parameter lists.....	9
Parameters.....	9
System control offsets.....	12
Support for multiple AMODES in a single process.....	12
Support for SRB callers.....	12
<b>Chapter 2. Callable services descriptions.....</b>	<b>13</b>
accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket.....	13
accept4 (BPX1AC4, BPX4AC4) — Accept a connection request from a client socket and set flags .....	16
accept_and_recv (BPX1ANR, BPX4ANR) — Accept a connection and receive the first block of data....	19
access (BPX1ACC, BPX4ACC) — Determine if a file can be accessed.....	24

aio_suspend (BPX1ASP, BPX4ASP) — Wait for an asynchronous I/O request.....	27
alarm (BPX1ALR, BPX4ALR) — Set an alarm .....	30
asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets.....	32
attach_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program .....	48
attach_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program .....	58
auth_check_resource_np (BPX1ACK, BPX4ACK) — Determine a user's access to a RACF- protected resource.....	64
__authenticate (BPX1AUT, BPX4AUT) — Authenticate the specified user's credentials.....	69
bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor.....	76
bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address.....	79
chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory .....	82
chatrat (BPX1CRA, BPX4CRA) — Change the attributes of a file or directory relative to a directory file descriptor.....	91
chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path .....	95
chdir (BPX1CHD, BPX4CHD) — Change the working directory .....	98
chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory .....	101
chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory.....	104
chpriority (BPX1CHP, BPX4CHP) — Change the scheduling priority of a process.....	107
chroot (BPX1CRT, BPX4CRT) — Change the root directory.....	111
clone (BPX1CLN, BPX4CLN) — Create a child process.....	114
close (BPX1CLO, BPX4CLO) — Close a file .....	121
closedir (BPX1CLD, BPX4CLD) — Close a directory.....	124
cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events.....	125
cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event .....	127
cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications .....	130
cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event .....	132
cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event .....	136
connect (BPX1CON, BPX4CON) — Establish a connection between two sockets.....	139
__console() (BPX1CCS, BPX4CCS) — Communicate with the console .....	142
__cpl (BPX1CPL) — CPL interface service.....	146
deletehfs (BPX1DEL, BPX4DEL) — Delete a program from storage .....	148
epoll_create (BPX1EPC, BPX4EPC) — Open an epoll file descriptor .....	150
epoll_ctl (BPX1EPL, BPX4EPL) — Control interface for epoll file descriptor.....	152
epoll_wait (BPX1EPW, BPX4EPW) — Wait for an I/O event on an epoll file descriptor.....	157
eventfd (BPX1EFD, BPX4EFD) — Create a file descriptor for event notification .....	160
exec (BPX1EXC, BPX4EXC) — Run a program.....	163
execmvs (BPX1EXM, BPX4EXM) — Run an MVS program .....	175
_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup .....	180
extlink_np (BPX1EXT, BPX4EXT) — Create an external symbolic link .....	183
faccessat (BPX1FAA, BPX4FAA) — Checks user's permissions of a file relative to a directory file descriptor.....	186
fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor.....	190
fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor.....	199
fchdir (BPX1FCD, BPX4FCD) — Change the working directory .....	202
fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor.....	204
fchmodat (BPX1CMA, BPX4CMA) — Change the mode of a file or directory relative to a directory file descriptor.....	206
fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor.....	210
fchownat (BPX1COA, BPX4COA) — Update the file ownership of a file or directory relative to a directory file descriptor .....	213
fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors.....	216
flock (BPX1FLK, BPX4FLK) — Apply or remove an advisory lock on an open file.....	225
fork (BPX1FRK, BPX4FRK) — Create a new process .....	227
fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables.....	233
freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures.....	236
fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor .....	238
fstatat (BPX1FSA, BPX4FSA) — Get file status relative to a directory file descriptor .....	241
fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status.....	244

fsync (BPX1FSY, BPX4FSY) — Write changes to permanent storage.....	247
ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file.....	249
getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location.....	251
getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier.....	258
getcwd (BPX1GCW, BPX4GCW) — Get the path name of the working directory.....	260
getegid (BPX1GEG, BPX4GEG) — Get the effective group ID.....	263
geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID.....	264
getgid (BPX1GID, BPX4GID) — Get the real group ID .....	265
getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database .....	266
getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID .....	268
getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name.....	271
getgroups (BPX1GGR, BPX4GGR) — Get a list of supplementary group IDs.....	274
getgroupsbyname (BPX1GUG, BPX4GUG) — Get a list of supplementary group IDs by user name....	276
gethostbyaddr (BPX1GHA, BPX4GHA) Get the IP address and alias of a host name for the specified IP address.....	279
gethostbyname (BPX1GHN, BPX4GHN) Get IP information for specified host domain names.....	282
gethostid or gethostname (BPX1HST, BPX4HST) — Get ID or name information about a socket host	285
getitimer (BPX1GTR, BPX4GTR) — Get the value of the interval timer.....	287
getlogin (BPX1GLG, BPX4GLG) — Get the user login name.....	289
getpeername or getsockname (BPX1GNM, BPX4GNM) — Get the name of a socket or of the peer connected to a socket.....	291
getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address..	291
getpgid (BPX1GEP, BPX4GEP) — Get the process group ID.....	295
getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID.....	297
getpid (BPX1GPI, BPX4GPI) — Get the process ID.....	298
getppid (BPX1GPP, BPX4GPP) — Get the parent process ID .....	299
getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process .....	300
getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database.....	302
getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name .....	305
getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID .....	308
getrandom (BPX1GRD, BPX4GRD) — Obtain a series of random bytes.....	310
getrlimit (BPX1GRL, BPX4GRL) — Get resource limits .....	313
getrusage (BPX1GRU, BPX4GRU) — Get resource usage .....	315
getsid (BPX1GES, BPX4GES) — Get the process group ID of the session leader.....	317
getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer.....	319
getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket.....	322
__getthent (BPX1GTH, BPX4GTH) — Get thread data.....	325
getuid (BPX1GUI, BPX4GUI) — Get the real user ID .....	329
getwd (BPX1GWD, BPX4GWD) — Get the path name of the working directory.....	330
givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program .....	332
grantpt (BPX1GPT, BPX4GPT) — Grant access to the subsidiary pseudoterminal.....	336
inotify_add_watch (BPX1IAW, BPX4IAW) — Add or modify a watch to an initialized inotify instance.	338
inotify_init (BPX1INI, BPX4INI) — Initialize an inotify instance.....	341
inotify_rm_watch (BPX1IRW, BPX4IRW) — Remove existing watches from inotify instances.....	344
IPCSDumpAccess (BPXGMPTR, BPXGMPT4) — PTRACE IPCS dump access service.....	346
IPCSDumpOpenClose (BPXGMCDE, BPXGMCD4) — MVS IPCS dump open/close service.....	352
isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal.....	357
isatty (BPX2ITY, BPX4ITY) (X/Open Version) — Determine whether a file descriptor represents a terminal.....	359
kill (BPX1KIL, BPX4KIL) — Send a signal to a process .....	360
__login, __login__applid, __certificate (BPX1SEC, BPX4SEC) — Provides an interface to the security product.....	364
lchattr (BPX1LCR, BPX4LCR) — Change the attributes of a file or directory or symbolic link.....	371
lchown (BPX1LCO, BPX4LCO) — Change the owner or group of a file, directory, or symbolic link.....	380
link (BPX1LNK, BPX4LNK) — Create a link to a file .....	383
linkat (BPX1LNA, BPX4LNA) — Make a link using at directory relative to a directory file descriptor ...	387

listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients .....	391
loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name.....	393
loadhfs extended (BPX1LDX, BPX4LDX) — Direct the loading of an executable into storage.....	399
lseek (BPX1LSK, BPX4LSK) — Change a file's offset .....	406
lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name.....	409
__map_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area.....	412
__map_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services.....	416
mkdir (BPX1MKD, BPX4MKD) — Make a directory .....	420
mkdirat (BPX1MDA, BPX4MDA) — Create a directory relative to a directory file descriptor .....	423
mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file.....	427
mknodat (BPX1MNA, BPX4MNA) — Make a directory, a FIFO, a character special, or a regular file relative to a directory file descriptor.....	431
mmap (BPX1MMP, BPX4MMP) — Map pages of memory .....	435
mount (BPX1MNT) — Make a file system available .....	444
__mount (BPX2MNT, BPX4MNT) — Make a file system available .....	447
mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping .....	452
msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations.....	455
msgget (BPX1QGT, BPX4QGT) — Create or find a message queue.....	458
msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue .....	462
msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue .....	465
msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage .....	469
munmap (BPX1MUN, BPX4MUN) — Unmap previously mapped addresses .....	473
mvsiptaffinity (BPX1IPT, BPX4IPT) — Run a program on the IPT thread .....	476
mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals.....	479
mvspauseinit (BPX1MPI, BPX4MPI) — Set up to wait on user events plus signals .....	481
mvspocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources .....	484
mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals.....	487
MVSThreadAffinity (BPX1TAF, BPX4TAF) — MVS thread affinity service.....	492
mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup .....	495
nice (BPX1NIC, BPX4NIC) — Change the nice value of a process.....	497
oe_env_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute.....	500
open (BPX1OPN, BPX4OPN) — Open a file.....	511
openat/openat2 (BPX1OAT, BPX4OAT) — Open a file with path name parameters relative to a directory file descriptor .....	518
opendir (BPX1OPD, BPX4OPD) — Open a directory.....	523
openstat (BPX2OPN, BPX4OPS) — Open a file and obtain status information.....	526
__passwd, __passwd__applid (BPX1PWD, BPX4PWD) — Verify or change security information.....	530
pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name..	535
pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal.....	539
pfsctl (BPX1PCT, BPX4PCT) — Physical file system control.....	541
__pid_affinity (BPX1PAF, BPX4PAF) — Add or delete an entry in a process's affinity list.....	548
pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe.....	551
pipe2 (BPX1PI2, BPX4PI2) — Create a new pipe instance .....	553
pivot_root (BPX1PVR, BPX4PVR) — Change the root mount.....	556
__poe() (BPX1POE, BPX4POE) — Port of entry information.....	559
poll (BPX1POL, BPX4POL) — Monitor activity on file descriptors and message queues.....	564
prctl (BPX1PRC, BPX4PRC) — Managing processes and threads .....	568
pread() and pwrite() (BPX1RW, BPX4RW) — Read from or write to a file without changing the file pointer .....	572
prlimit (BPX1PRL, BPX4PRL) — Get and set resource limits for specified processes .....	574
pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread.....	579
pthread_create (BPX1PTC, BPX4PTC) — Create a thread.....	581
pthread_detach (BPX1PTD, BPX4PTD) — Detach a thread.....	586
pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread .....	588
pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread.....	592
pthread_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread.....	595

pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process.....	598
pthread_security_np, pthread_security_applid_np (BPX1TLS, BPX4TLS) — Create or delete thread-level security.....	601
pthread_self (BPX1PTS, BPX4PTS) — Query the thread ID .....	609
pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state.....	610
pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type.....	613
pthread_tag_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data .....	615
pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur.....	618
ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging.....	620
querydub (BPX1QDB, BPX4QDB) — Obtain the dub status of the current task .....	646
queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered.....	648
quiesce (BPX1QSE, BPX4QSE) — Quiesce a file system.....	650
read (BPX1RED, BPX4RED) — Read from a file or socket.....	652
readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory .....	656
readdir2 (BPX1RD2, BPX4RD2) — Read an entry from a directory.....	659
read_extlink (BPX1RDX, BPX4RDX) — Read an external symbolic link .....	663
readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link .....	666
readlinkat (BPX1RLA, BPX4RLA) — Read the value of a symbolic link relative to a directory file descriptor.....	669
readv (BPX1RDV, BPX4RDV) — Read data and store it in a set of buffers .....	673
realpath (BPX1RPH, BPX4RPH) — Resolve a path name.....	677
recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer.....	680
recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer.....	683
recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers .....	687
rename (BPX1REN, BPX4REN) — Rename a file or directory .....	690
renameat (BPX1RNA, BPX4RNA) — Rename a file using the at directory relative to a directory file descriptor .....	694
resource (BPX1RMG, BPX4RMG) — Measure resources .....	699
rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning .....	701
rmdir (BPX1RMD, BPX4RMD) — Remove a directory .....	703
select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues .....	706
semctl (BPX1SCT, BPX4SCT) — Perform semaphore control operations .....	713
semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores.....	718
semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations.....	723
send (BPX1SND, BPX4SND) — Send data on a socket .....	726
send_file (BPX1SF, BPX4SF) — Send a file on a socket.....	730
sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket .....	734
sendto (BPX1STO, BPX4STO) — Send data on a socket .....	738
server_init (BPX1SIN, BPX4SIN) — Server initialization .....	742
server_pwu (BPX1SPW, BPX4SPW) — Server process work unit .....	746
set_dub_default (BPX1SDD, BPX4SDD) — Set the dub default service .....	752
setegid (BPX1SEG, BPX4SEG) — Set the effective group ID .....	758
seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID.....	760
setgid (BPX1SGI, BPX4SGI) — Set the group ID.....	762
setgrent (BPX1SGE, BPX4SGE) — Reset the group database.....	765
setgroups (BPX1SGR, BPX4SGR) — Set the supplementary group IDs list .....	766
setitimer (BPX1STR, BPX4STR) — Set the value of the interval timer .....	768
setns (BPX1SNS, BPX4SNS) — Reassociate thread with a namespace.....	771
setpeer (BPX1SPR, BPX4SPR) — Preset the peer address associated with a socket .....	775
setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control .....	777
setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process.....	779
setpwent (BPX1SPE, BPX4SPE) — Reset the user database .....	782
setregid (BPX1SRG, BPX4SRG) — Set the real and effective GIDs .....	784
setreuid (BPX1SRU, BPX4SRU) —Set the real and effective UIDs.....	786
setrlimit (BPX1SRL, BPX4SRL) — Set resource limits .....	789
setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID .....	793

setsockopt or getsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket.....	795
set_thread_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread_created threads.....	795
set_timer_event (BPX1STE, BPX4STE) — Set DIE-mode timer event.....	798
setuid (BPX1SUI, BPX4SUI) — Set user IDs.....	800
shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment .....	804
shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations .....	809
shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment.....	812
shmlock (BPX1SLK, BPX4SLK) — Shared memory lock service.....	814
shmlock_condvar (BPX1SMC, BPX4SMC) — Shared mutex and condition variable service.....	819
shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment.....	827
shutdown (BPX1SHT, BPX4SHT) — Shut down all or part of a duplex socket connection .....	832
sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action .....	834
__sigactionset (BPX1SA2, BPX4SA2) — Examine or change a set of signal actions.....	839
sigpending (BPX1SIP, BPX4SIP) — Examine pending signals .....	843
sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask .....	845
sigqueue (BPX1SGQ, BPX4SGQ) — Queue a signal to a process.....	848
sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered.....	851
sigtimedwait (BPX1STW, BPX4STW) — Wait for a signal with a specified timeout.....	854
sigwait (BPX1SWT, BPX4SWT) — Wait for a signal .....	857
sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time .....	859
smf_record (BPX1SMF, BPX4SMF) — Write an SMF record .....	861
socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets .....	865
spawn (BPX1SPN, BPX4SPN) — Spawn a process .....	869
srx_np (BPX1SRX, BPX4SRX) — Send or receive CSM buffers on a socket .....	886
stat (BPX1STA, BPX4STA) — Get status information about a file by path name .....	893
statvfs (BPX1STV, BPX4STV) — Get the file system status .....	896
sw_sigldlv (BPX1DSD, BPX4DSD) — Switch the setting for signal delivery.....	899
symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name .....	900
symlinkat (BPX1SYA, BPX4SYA) — Create a symbolic link relative to a directory file descriptor .....	905
sync (BPX1SYN, BPX4SYN) — Schedule file system updates .....	910
syncfs (BPX1SYF, BPX4SYF) — Synchronizes the file system file referred to by the open file descriptor .....	912
sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options .....	913
takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program .....	916
tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted .....	918
tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal .....	921
tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal .....	923
tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal .....	926
tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names .....	928
tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID .....	931
tcgetsid (BPX1TGS, BPX4TGS) — Get a process group ID for the session leader for the controlling terminal .....	933
tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal .....	934
tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal .....	937
tcsetcp (BPX1TSC, BPX4TSC) — Set terminal code page names .....	940
tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID .....	944
tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables.....	946
times (BPX1TIM, BPX4TIM) — Get process and child process times .....	950
truncate (BPX1TRU, BPX4TRU) — Change the size of a file .....	953
ttynname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal .....	956
ttynname (BPX2TYN, BPX4TYN) (X/Open version) — Get the name of a terminal.....	958
umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask.....	960
umount (BPX1UMT, BPX4UMT) — Remove a virtual file system .....	961
uname (BPX1UNA, BPX4UNA) — Obtain the name of the current operating system .....	964
unlink (BPX1UNL, BPX4UNL) — Remove a directory entry .....	966



unlinkat (BPX1ULA, BPX4ULA) — Remove a directory or directory entry relative to a directory file descriptor.....	969
unlockpt (BPX1UPT, BPX4UPT) — Unlock a pseudoterminal manager and subsidiary pair .....	973
unquiesce (BPX1UQS, BPX4UQS) — Unquiesce a file system .....	975
unshare (BPX1UNS, BPX4UNS) — Run program in a new namespace.....	977
utime (BPX1UTI, BPX4UTI) — Set file access and modification times .....	981
wait (BPX1WAT, BPX4WAT) — Wait for a child process to end .....	984
wait-extension (BPX1WTE, BPX4WTE) — Obtain status information for children.....	987
w_getipc (BPX1GET, BPX4GET) — Query interprocess communications.....	991
w_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems .....	995
w_getpsent (BPX1GPS) — Get process data .....	999
w_ioctl (BPX1IOC, BPX4IOC) — Control I/O .....	1003
__wlm (BPX1WLM, BPX4WLM) — WLM interface service .....	1018
w_pioctl (BPX1PIO, BPX4PIO) — Path name I/O control.....	1025
w_statvfs (BPX1STF, BPX4STF) — Get the file system status .....	1029
write (BPX1WRT, BPX4WRT) — Write to a file or a socket.....	1032
writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers .....	1036
<b>Chapter 3. Callable services for SRB mode routines.....</b>	<b>1041</b>
Overview of callable services available to SRB mode routines.....	1041
Recovery.....	1041
Task mode routine responsibilities.....	1041
Task and address space dynamic resource manager.....	1042
Callable services supported in SRB mode.....	1042
<b>Appendix A. System control offsets to callable services.....</b>	<b>1045</b>
Example.....	1045
List of offsets.....	1045
<b>Appendix B. Mapping macros—AMODE 31.....</b>	<b>1055</b>
Macros mapping parameter options.....	1055
BPXYACC — Map flag values for access.....	1055
BPXYAIO — Map asyncio parameter list.....	1055
BPXYATT — Map file attributes for chattr and fchattr.....	1057
BPXYAUDT — Map flag values for chaudit and fchaudit.....	1060
BPXYBRLK — Map byte range lock request for fcntl.....	1060
BPXYCCA — Map input/output structure for __console().....	1060
BPXYCID — Map the returning structure for getclientid().....	1061
BPXYCLNP— Map clone syscall parameters.....	1062
BPXYCONS — Constants used by services.....	1062
BPXYCW — Serialization constants used by many services.....	1069
BPXYDCOR — dbx cordump cache information.....	1069
BPXYDIRE — Map directory entries for readdir.....	1074
BPXYENFO — ENF signal constants.....	1075
BPXYEFD— Map eventfd syscall parameters.....	1075
BPXYEPOL— Map epoll syscall parameters.....	1075
BPXYERNO — Component return and reason codes.....	1076
BPXYFCTL — Command values and flags for fcntl.....	1076
BPXYFDUM — Logical file system dump parameter list.....	1077
BPXYFLK— Applying or removing advisory locks.....	1078
BPXYFTYP — File type definitions.....	1078
BPXYFUIO — Map file system user I/O block.....	1078
BPXYGIDN — Map data returned for getpwnam and getpwuid.....	1079
BPXYGIDS — Map data returned for getgrnam and getgrpid.....	1080
BPXYGRD — Map the response structure for getrandom.....	1080
BPXYINFY— Map inotify_init syscall parameters.....	1080
BPXYINHE — Spawn Inheritance Structure.....	1084

BPXYIOCC — Ioctl command definitions.....	1085
BPXYIOC6 — Map IPV6 prerouter structures.....	1094
BPXYIOV — Map the I/O vector structure.....	1098
BPXYIPCP — Map interprocess communication permissions.....	1098
BPXYIPCQ — Map w_getipc structure.....	1098
BPXYITIM — Map getitimer, setitimer structure .....	1101
BPXYMMG — Map interface for _map_init and _map_service .....	1101
BPXYMNTE — Map response and element structure of w_getmntent.....	1104
BPXYMODE — Map the mode constants .....	1106
BPXYMSG — Map interprocess communication message queues.....	1107
BPXYMSGF — Map the message flags.....	1107
BPXYMSGH — Map the message header.....	1109
BPXYMSGX — Map the message header .....	1109
BPXYMTM — Map the modes for mount and unmount .....	1110
BPXYOCRT — Map the OE certificate support structure .....	1111
BPXYOEXT — Map the common external control block.....	1112
BPXYOPNF — Map flag values for open.....	1113
BPXYPCF — Command values for pathconf and pathconf.....	1114
BPXPEDB — Mapping of process exit data block.....	1114
BPXYPGPS — Map the response structure for w_getpsent .....	1116
BPXYPGTH — Map the __getthent input/output structure.....	1119
BPXYPOE — Map poe syscall parameters.....	1123
BPXPOLL — Map poll syscall parameters.....	1124
BPXYPPSD — Map signal delivery data.....	1124
BPXPRLI — Process-level information.....	1126
BPXYPTAT — Map attributes for pthread_exit_and_get.....	1127
BPXPTRC — Map parameters for ptrace.....	1127
BPXYPTXL — Map the parameter list for pthread_create.....	1141
BPXYRFIS — Map the register file interest structures.....	1141
BPXYRLIM — Map the rlimit, rusage, and timeval structures.....	1142
BPXYRMON — Map resource monitor data.....	1142
BPXYSECI — Map the output of BPX1IOC for the SECIGET request.....	1144
BPXYSECO — Map the input/output of BPX1IOC for the SIOCSECENVR request.....	1144
BPXYSECT — Map the output of BPX1IOC for the SECIGET_T request.....	1144
BPXYSEEK — Constants for lseek.....	1144
BPXYSEL — Map the select options.....	1145
BPXYSELT — Map the timeout value for the select syscall .....	1145
BPXYSEM — Map interprocess communication semaphores.....	1146
BPXYSFDL — Map the server file descriptor list structure.....	1146
BPXYSFPL — Map the send_file parameter list.....	1147
BPXYSHM—Map interprocess communication shared memory segments.....	1147
BPXYSIGH — Signal constants.....	1148
BPXYSINF — Map SIGINFO_T structure.....	1150
BPXYSMC — Map shared mutex/condvar declares and constants.....	1150
BPXYSOCK — Map SOCKADDR structure and constants.....	1151
BPXYSSET — Map the sigaction set.....	1160
BPXYSSTF — Map response structure for file system status.....	1161
BPXYSTAT — Map the response structure for stat .....	1162
BPXYTCCP — Map the terminal control code page structure.....	1163
BPXYTHDQ — Mapping of THDQ structure for BPX1PQG.....	1164
BPXYTHLI — Thread-level information.....	1165
BPXYTIMS — Map the response structure for times.....	1168
BPXYTIOS — Map the termios structure.....	1169
BPXYUTSN — Map the response structure for uname.....	1172
BPXYWAST — Map the wait status word.....	1172
BPXYWLM — WLM constants and parameter list DSECTs.....	1173
BPXYWNSZ — Map the winsize structure.....	1179
BPXZOAPB — z/OS UNIX address space per-process extension.....	1179

BPXZOCVT — Base control block for z/OS UNIX.....	1180
BPXZOTCB — z/OS UNIX extension to the TCB.....	1181

## **Appendix C. Mapping macros—AMODE 64.....1187**

Macros mapping parameter options.....	1187
BPXYAIO — Map asyncio parameter list.....	1187
BPXYCCA — Map input/output structure for __console().....	1189
BPXYDCOR — dbx cordump cache information.....	1190
BPXYINHE — Spawn inheritance structure.....	1195
BPXYIOC6 — Map IPV6 prerouter structures.....	1196
BPXYIOV — Map the I/O vector structure.....	1199
BPXYIPCQ — Map w_getipc structure.....	1199
BPXYITIM — Map getitimer, setitimer structure.....	1202
BPXYMMG — Map Interface for _map_init and _map_service.....	1202
BPXYMSG — Map interprocess communication message queues.....	1204
BPXYMSGH — Map the message header.....	1205
BPXYOCRT — Map the OE certificate support structure.....	1206
BPXYPPSD — Map signal delivery data.....	1206
BPXYPTXL — Map the parameter pist for pthread_create.....	1208
BPXYRLIM — Map the rlimit, rusage, and timeval structures.....	1208
BPXYSELT — Map the timeout value for the select syscall.....	1209
BPXYSEM — Map interprocess communication semaphores.....	1209
BPXYSFPL — Map the send_file parameter list.....	1210
BPXYSHM—Map interprocess communication shared memory segments.....	1210
BPXYSINF — Map SIGINFO_T structure.....	1211
BPXYSSET — Map the sigaction set.....	1211
BPXYWLM — WLM constants and parameter list DSECTs.....	1212

## **Appendix D. Callable services examples—AMODE 31..... 1219**

Reentrant entry linkage.....	1219
BPX1ACC (access) example.....	1219
BPX1AC4 (accept4) example.....	1220
BPX1ACK (auth_check_resource_np) example.....	1220
BPX1ACP (accept) example.....	1220
BPX1AC4 (accept4) example.....	1221
BPX1AIO (asyncio) example.....	1221
BPX1ALR (alarm) example.....	1221
BPX1ANR (accept_and_recv) example.....	1222
BPX1ASP (aio_suspend) example.....	1222
BPX1ATM (attach_execmvs) example.....	1222
BPX1ATX (attach_exec) example.....	1223
BPX1AUT (__authenticate) example.....	1223
BPX1BND (bind) example.....	1224
BPX1BAS (bind with source address selection) example.....	1224
BPX1CCA (cond_cancel) example.....	1225
BPX1CCS (__console()) example.....	1225
BPX1CHA (chaudit) example.....	1225
BPX1CHD (chdir) example.....	1226
BPX1CHM (chmod) example.....	1226
BPX1CHO (chown) example.....	1226
BPX1CHOA (fchownat) example.....	1226
BPX1CHP (chpriority) example.....	1227
BPX1CMA (fchmodat) example.....	1227
BPX1CHR (chattr) example.....	1228
BPX1CLN (clone) example.....	1228
BPX1CLD (closedir) example.....	1229
BPX1CLO (close) example.....	1229

BPX1CON (connect) example.....	1229
BPX1CPO (cond_post) example.....	1230
BPX1CRA (chatrat) example.....	1230
BPX1CRT (chroot) example.....	1230
BPX1CRT (chroot) example.....	1231
BPX1CSE (cond_setup) example.....	1231
BPX1CTW (cond_timed_wait) example.....	1231
BPX1CWA (cond_wait) example.....	1231
BPX1DEL (deleteHFS) example.....	1232
BPX1EFD (eventfd) example.....	1232
BPX1ENV (oe_env_np) example.....	1233
BPX1EPC (epoll_create) example.....	1233
BPX1EPL (epoll_epl) example.....	1233
BPX1EPW (epoll_wait) example.....	1234
BPX1EXC (exec) example.....	1234
BPX1EXI (_exit) example.....	1235
BPX1EXM (execmvs) example.....	1235
BPX1EXT (extlink_np) example.....	1235
BPX1FAA (faccessat) example.....	1235
BPX1FAI (freeaddrinfo) example.....	1236
BPX1FCA (fchaudit) example.....	1236
BPX1FCD (fchdir) example.....	1237
BPX1FCM (fchmod) example.....	1237
BPX1FCO (fchown) example.....	1237
BPX1FCR (fchatrr) example.....	1237
BPX1FCT (fcntl) example.....	1238
BPX1FLK (flock) example.....	1239
BPX1FPC (fpathconf) example.....	1239
BPX1FRK (fork) example.....	1239
BPX1FST (fstat) example.....	1239
BPX1FSA (fstatat) example.....	1240
BPX1FSY (fsync) example.....	1240
BPX1FTR (ftruncate) example.....	1240
BPX1FTV (fstatvfs) example.....	1240
BPX1GAI (getaddrinfo) example .....	1241
BPX1GCL (getclientid) example.....	1241
BPX1GCW (getcwd) example .....	1241
BPX1GEG (getegid) example .....	1242
BPX1GEP (getpgid) example.....	1242
BPX1GES (getsid) example.....	1242
BPX1GET (w_getipc) example.....	1242
BPX1GEU (geteuid) example.....	1243
BPX1GGE (getgrent) example.....	1243
BPX1GGI (getgrgid) example.....	1243
BPX1GGN (getgrnam) example .....	1244
BPX1GGR (getgroups) example .....	1244
BPX1GHA (gethostbyaddr) example.....	1244
BPX1GHN (gethostbyname) example .....	1245
BPX1GID (getgid) example.....	1245
BPX1GIV (givesocket) example .....	1245
BPX1GLG (getlogin) example.....	1246
BPX1GMN (w_getmntent) example.....	1246
BPX1GNI (getnameinfo) example.....	1246
BPX1GPG (getpgrp) example.....	1247
BPX1GNM (getpeername or getsockname) example.....	1247
BPX1GPE (getpwent) example.....	1247
BPX1GPI (getpid) example.....	1248
BPX1GPN (getpwnam) example .....	1248

BPX1GPP (getppid) example.....	1248
BPX1GPS (w_getpsent) example .....	1248
BPX1GPT (grantpt) example.....	1249
BPX1GPU (getpwuid) example.....	1249
BPX1GPY (getpriority) example.....	1249
BPX1GRD (getrandom) example.....	1250
BPX1GRL (getrlimit) example.....	1250
BPX1GRU (getrusage) example.....	1250
BPX1GTH (__getthent) example.....	1251
BPX1GTR (getitimer) example.....	1251
BPX1GUG (getgroupsbyname) example.....	1251
BPX1GUI (getuid) example.....	1252
BPX1GWD (getwd) example.....	1252
BPX1HST (gethostid or gethostname) example.....	1252
BPX1IAW (inotify_add_watch) example.....	1252
BPX1INI (inotify_init) example.....	1253
BPX1IOC (w_iocctl) example.....	1253
BPX1IPT (mvsiptaffinity) example.....	1253
BPX1IRM (inotify_rm_watch) example.....	1254
BPX1ITY (isatty) example.....	1254
BPX2ITY (isatty) example .....	1254
BPX1KIL (kill) example.....	1254
BPX1LCO (lchown) example.....	1255
BPX1LCR (lchattr) example.....	1255
BPX1LDX (loadhfs extended) example.....	1255
BPX1LOD (loadhfs) example.....	1257
BPX1LNA (linkat) example.....	1257
BPX1LNK (link) example.....	1258
BPX1LSK (lseek) example.....	1258
BPX1LSN (listen) example.....	1258
BPX1LST (lstat) example.....	1258
BPX1MAT (shmat) example.....	1259
BPX1MCT (shmctl) example.....	1259
BPX1MDA (mkdirat) example.....	1259
BPX1MDT (shmdt) example.....	1260
BPX1MGT (shmget) example.....	1260
BPX1MKD (mkdir) example.....	1260
BPX1MKN (mknod) example.....	1261
BPX1MMI (__map_init) example.....	1261
BPX1MMP (mmap) example.....	1262
BPX1MMS (__map_service) example.....	1262
BPX1MNA (mknodat) example.....	1262
BPX1MNT (mount) example.....	1263
BPX2MNT (__mount) example.....	1263
BPX1MP (mvspause) example.....	1263
BPX1MPC (mvspocclp) examples.....	1264
BPX1MPI (mvspauseinit) example.....	1265
BPX1MPR (mprotect) example.....	1265
BPX1MSD (mvsunsigsetup) example.....	1265
BPX1MSS (mvssigsetup) example.....	1266
BPX1MSY (msync) example.....	1266
BPX1MUN (munmap) example.....	1266
BPX1NIC (nice) example.....	1266
BPX1OPD (opendir) example.....	1267
BPX1OPN (open) example.....	1267
BPX2OPN (openstat) example.....	1267
BPX1OPT (getsockopt or setsockopt) example.....	1268
BPX1PAF (__pid_affinity) example.....	1268

BPX1PAS (pause) example.....	1269
BPX1PCF (pathconf) example.....	1269
BPX1PCT (pfsctl) example.....	1269
BPX1PIP (pipe) example.....	1270
BPX1PI2 (pipe2) example.....	1270
BPX1POE (__poe) example.....	1270
BPX1POL (poll) example.....	1270
BPX1PRC (prctl) example.....	1271
BPX1PRL (prlimit) example.....	1271
BPX1PSI (pthread_setintr) example .....	1272
BPX1PST (pthread_setintrtype) example.....	1272
BPX1PTB (pthread_cancel) example .....	1272
BPX1PTC (pthread_create) example.....	1272
BPX1PTD (pthread_detach) example .....	1273
BPX1PTI (pthread_testintr) example.....	1273
BPX1PTJ (pthread_join) example.....	1273
BPX1PTK (pthread_kill) example.....	1274
BPX1PTQ (pthread_quiesce) example .....	1274
BPX1PTR (ptrace) example.....	1274
BPX1PTS (pthread_self) example.....	1274
BPX1PTT (pthread_tag_np) example.....	1275
BPX1PTX (pthread_exit_and_get) example .....	1275
BPX1PVR (pivot_root) example.....	1275
BPX1PWD (__passwd, __passwd__applid) example.....	1276
BPX1QCT (msgctl) example.....	1276
BPX1QDB (querydub) example.....	1276
BPX1QGT (msgget) example.....	1276
BPX1QRC (msgrcv) example.....	1277
BPX1QSE (quiesce) example.....	1277
BPX1QSN (msgsnd) example .....	1277
BPX1RCV (recv) example.....	1278
BPX1RDD (readdir) example.....	1278
BPX1RDL (readlink) example.....	1278
BPX1RDV (readv) example.....	1279
BPX1RDX (read extlink) example.....	1279
BPX1RD2 (readdir2) example.....	1279
BPX1RED (read) example.....	1280
BPX1REN (rename) example.....	1280
BPX1RFM (recvfrom) example.....	1280
BPX1RLA (readlinkat) example.....	1281
BPX1RMD (rmdir) example.....	1281
BPX1RMG (resource) example.....	1281
BPX2RMS (recvmsg) example.....	1282
BPX1RNA (renameat) example.....	1282
BPX1RPH (realpath) example.....	1283
BPX1RW (Pwrite) example.....	1283
BPX1RWD (rewinddir) example.....	1283
BPX1SA2 (__sigactionset) example.....	1284
BPX1SCT (semctl) example.....	1284
BPX1SDD (setdubdefault) example.....	1284
BPX1SEC (__login, __login__applid, __certificate) example.....	1285
BPX1SEG (setegid) example.....	1285
BPX1SEL (select) example.....	1285
BPX1SEU (seteuid) example.....	1286
BPX1SF (send_file) example.....	1286
BPX1SGE (setgrent) example.....	1286
BPX1SGI (setgid) example.....	1287
BPX1SGQ (sigqueue) example.....	1287

BPX1SGR (setgroups) example.....	1287
BPX1SGT (semget) example.....	1287
BPX1SHT (shutdown) example.....	1288
BPX1SIA (sigaction) example.....	1288
BPX1SIN (server_init) example.....	1288
BPX1SIP (sigpending) example.....	1289
BPX1SLK (shmem_lock) example.....	1289
BPX1SLP (sleep) example.....	1289
BPX1SMF (smf_record) example.....	1290
BPX2SMS (sendmsg) example.....	1290
BPX1SND (send) example.....	1291
BPX1SNS (setns) example.....	1291
BPX1SOC (socket or socketpair) example.....	1291
BPX1SOP (semop) example.....	1292
BPX1SPB (queue_interrupt) example.....	1292
BPX1SPE (setpwent) example.....	1292
BPX1SPG (setpgid) example.....	1293
BPX1SPM (sigprocmask) example.....	1293
BPX1SPN (spawn) example.....	1293
BPX1SPR (setpeer) example.....	1294
BPX1SPW (server_pwu) example.....	1294
BPX1SPY (setpriority) example.....	1295
BPX1SRG (setregid) example.....	1295
BPX1SRL (setrlimit) example.....	1295
BPX1SRU (setreuid) example.....	1296
BPX1SRX (srx_np) example.....	1296
BPX1SSI (setsid) example.....	1296
BPX1SSU (sigsuspend) example.....	1297
BPX1STA (stat) example.....	1297
BPX1STE (set_timer_event) example.....	1297
BPX1STF (w_statvfs) example.....	1297
BPX1STL (set_thread_limits) example.....	1298
BPX1STO (sendto) example.....	1298
BPX1STR (setitimer) example.....	1298
BPX1STV (statvfs) example.....	1299
BPX1STW (sigtimedwait) example.....	1299
BPX1SUI (setuid) example.....	1299
BPX1SWT (sigwait) example.....	1300
BPX1SYA (symlinkat) example.....	1300
BPX1SYC (sysconf) example.....	1300
BPX1SYF (syncfs) example.....	1300
BPX1SYM (symlink) example.....	1301
BPX1SYN (sync) example.....	1301
BPX1TAF (MVSThreadAffinity) example.....	1301
BPX1TAK (takesocket) example.....	1301
BPX1TDR (tcdrain) example.....	1302
BPX1TFH (tcflush) example.....	1302
BPX1TFW (tcflow) example.....	1302
BPX1TGA (tcgetattr) example.....	1302
BPX1TGC (tcgetcp) example.....	1303
BPX1TGP (tcgetpgrp) example.....	1303
BPX1TGS (tcgetsid) example.....	1303
BPX1TIM (times) example.....	1303
BPX1TLS (pthread_security_np) example.....	1304
BPX1TRU (truncate) example.....	1304
BPX1TSA (tcsetattr) example.....	1304
BPX1TSB (tcsendbreak) example.....	1305
BPX1TSC (tcsetcp) example.....	1305

BPX1TSP (tcsetpgrp) example.....	1305
BPX1TST (tcsettables) example.....	1305
BPX1TYN (ttyname) example.....	1306
BPX2TYN (ttyname) example.....	1306
BPX1ULA (unlinkat) example.....	1306
BPX1UMK (umask) example.....	1307
BPX1UMT (umount) example.....	1307
BPX1UNA (uname) example.....	1307
BPX1UNL (unlink) example.....	1308
BPX1UPT (unlockpt) example.....	1308
BPX1UQS (unquiesce) example .....	1308
BPX1UNS (unshare) example.....	1308
BPX1UTI (utime) example .....	1309
BPX1WAT (wait) example .....	1309
BPX1WLM (__WLM) example .....	1309
BPX1WRT (write) example.....	1310
BPX1WRV (writev) example.....	1310
BPX1WTE (wait extension) example.....	1310
Reentrant return linkage.....	1311

## **Appendix E. Callable services examples—AMODE 64.....1313**

Reentrant entry linkage.....	1313
BPX4ACC (access) example.....	1313
BPX4AC4 (accept4) example.....	1314
BPX4ACK (auth_check_resource_np) example.....	1314
BPX4ACP (accept) example.....	1314
BPX4AC4 (accept4) example.....	1315
BPX4AIO (asyncio) example.....	1315
BPX4ALR (alarm) example.....	1315
BPX4ANR (accept_and_recv) example.....	1316
BPX4ASP (aio_suspend) example.....	1316
BPX4ATM (attach_execmvs) example.....	1316
BPX4ATX (attach_exec) example.....	1317
BPX4AUT (__authenticate) example.....	1317
BPX4BND (bind) example.....	1318
BPX4BAS (bind with source address selection) example.....	1318
BPX4CCA (cond_cancel) example.....	1318
BPX4CCS (__console()) example.....	1319
BPX4CHA (chaudit) example.....	1319
BPX4CHD (chdir) example.....	1319
BPX4CHM (chmod) example.....	1319
BPX4CHO (chown) example.....	1320
BPX4CHOA (fchownat) example.....	1320
BPX4CHP (chpriority) example.....	1320
BPX4CMA (fchmodat) example.....	1321
BPX4CHR (chattr) example.....	1321
BPX4CLD (closedir) example.....	1322
BPX4CLN (clone) example.....	1322
BPX4CLO (close) example.....	1323
BPX4CON (connect) example.....	1323
BPX4CPO (cond_post) example.....	1323
BPX4CRA (chatrat) example.....	1323
BPX4CRT (chroot) example.....	1324
BPX4CSE (cond_setup) example.....	1324
BPX4CTW (cond_timed_wait) example.....	1325
BPX4CWA (cond_wait) example.....	1325
BPX4DEL (deleteHFS) example.....	1325



BPX4EFD (eventfd) example.....	1326
BPX4ENV (oe_env_np) example.....	1326
BPX4EPC (epoll_create) example.....	1326
BPX4EPL (epoll_epl ) example.....	1327
BPX4EPW (epoll_wait) example.....	1327
BPX4EXC (exec) example.....	1327
BPX4EXI (_exit) example.....	1328
BPX4EXM (execmvs) example.....	1328
BPX4EXT (extlink_np) example.....	1329
BPX4FAA (faccessat) example.....	1329
BPX4FAI (freeaddrinfo) example.....	1329
BPX4FCA (fchmod) example.....	1330
BPX4FCD (fchdir) example.....	1330
BPX4FCM (fchmod) example.....	1330
BPX4FCO (fchown) example.....	1330
BPX4FCR (fchatrr) example.....	1331
BPX4FCT (fcntl) example.....	1331
BPX4FLK (flock) example.....	1332
BPX4FPC (fpathconf) example.....	1332
BPX4FRK (fork) example.....	1332
BPX4FST (fstat) example.....	1333
BPX1FSA (fstatat) example.....	1333
BPX4FSY (fsync) example.....	1333
BPX4FTR (ftruncate) example.....	1334
BPX4FTV (fstatvfs) example.....	1334
BPX4GAI (getaddrinfo) example .....	1334
BPX4GCL (getclientid) example.....	1335
BPX4GCW (getcwd) example .....	1335
BPX4GEG (getegid) example .....	1335
BPX4GEP (getpgid) example.....	1335
BPX4GES (getsid) example.....	1336
BPX4GET (w_getipc) example.....	1336
BPX4GEU (geteuid) example.....	1336
BPX4GGE (getgrent) example.....	1336
BPX4GGI (getgrgid) example.....	1337
BPX4GGN (getgrnam) example .....	1337
BPX4GGR (getgroups) example.....	1338
BPX4GHA (gethostbyaddr) example.....	1338
BPX4GHN (gethostbyname) example.....	1338
BPX4GID (getgid) example.....	1339
BPX4GIV (givesocket) example.....	1339
BPX4GLG (getlogin) example.....	1339
BPX4GMN (w_getmntent) example.....	1340
BPX4GNI (getnameinfo) example.....	1340
BPX4GPG (getpgrp) example.....	1340
BPX4GNM (getpeername or getsockname) example.....	1340
BPX4GPE (getpwent) example.....	1341
BPX4GPI (getpid) example.....	1341
BPX4GPN (getpwnam) example.....	1341
BPX4GPP (getppid) example.....	1342
BPX4GPT (grantpt) example.....	1342
BPX4GPU (getpwuid) example.....	1342
BPX4GPY (getpriority) example.....	1342
BPX4GRD (getrandom) example.....	1343
BPX4GRL (getrlimit) example.....	1343
BPX4GRU (getrusage) example.....	1343
BPX4GTH (__getthent) example.....	1344
BPX4GTR (getitimer) example.....	1344

BPX4GUG (getgroupsbyname) example.....	1344
BPX4GUI (getuid) example.....	1345
BPX4GWD (getwd) example.....	1345
BPX4HST (gethostid or gethostname) example.....	1345
BPX4IAW (inotify_add_watch) example.....	1346
BPX4INI (inotify_init) example.....	1346
BPX4IRM (inotify_rm_watch) example.....	1346
BPX4IOC (w_iocctl) example.....	1347
BPX4IPT (mvsiptaffinity) example.....	1347
BPX4ITY (isatty) example.....	1347
BPX4KIL (kill) example.....	1347
BPX4LCO (lchown) example.....	1348
BPX4LCR (lchattr) example.....	1348
BPX4LDX (loadhfs extended) example.....	1348
BPX4LNA (linkat) example.....	1350
BPX4LOD (loadhfs) example.....	1350
BPX4LNK (link) example.....	1351
BPX4LSK (lseek) example.....	1351
BPX4LSN (listen) example.....	1351
BPX4LST (lstat) example.....	1351
BPX4MAT (shmat) example.....	1352
BPX4MCT (shmctl) example.....	1352
BPX4MDA (mkdirat) example.....	1352
BPX4MDT (shmdt) example.....	1353
BPX4MGT (shmget) example.....	1353
BPX4MKD (mkdir) example.....	1353
BPX4MKN (mknod) example.....	1354
BPX4MMI (__map_init) example.....	1354
BPX4MMP (mmap) example.....	1355
BPX4MMS (__map_service) example.....	1355
BPX4MNA (mknodat) example.....	1355
BPX4MNT (__mount) example.....	1356
BPX4MP (mvspause) example.....	1356
BPX4MPC (mvspocclp) example.....	1356
BPX4MPI (mvspauseinit) example.....	1357
BPX4MPR (mprotect) example.....	1357
BPX4MSD (mvsunsigsetup) example.....	1357
BPX4MSS (mvssigsetup) example.....	1358
BPX4MSY (msync) example.....	1358
BPX4MUN (munmap) example.....	1358
BPX4NIC (nice) example .....	1358
BPX4OPD (opendir) example.....	1359
BPX4OPN (open) example.....	1359
BPX4OPS (openstat) example.....	1359
BPX4OPT (getsockopt or setsockopt) example.....	1360
BPX4PAF (__pid_affinity) example.....	1360
BPX4PAS (pause) example.....	1361
BPX4PCF (pathconf) example.....	1361
BPX4PCT (pfsctl) example.....	1361
BPX4PIP (pipe) example.....	1362
BPX4PI2 (pipe2) example.....	1362
BPX4POE (__poe) example.....	1362
BPX4POL (poll) example.....	1362
BPX4PRC (prctl) example.....	1363
BPX4PRL (prlimit) example.....	1363
BPX4PSI (pthread_setintr) example.....	1364
BPX4PST (pthread_setintrtype) example.....	1364
BPX4PTB (pthread_cancel) example.....	1364

BPX4PTC (pthread_create) example.....	1364
BPX4PTD (pthread_detach) example.....	1365
BPX4PTI (pthread_testintr) example.....	1365
BPX4PTJ (pthread_join) example.....	1365
BPX4PTK (pthread_kill) example.....	1366
BPX4PTQ (pthread_quiesce) example.....	1366
BPX4PTR (ptrace) example.....	1366
BPX4PRL (prlimit) example.....	1366
BPX4PTS (pthread_self) example.....	1367
BPX4PTT (pthread_tag_np) example.....	1367
BPX4PTX (pthread_exit_and_get) example.....	1367
BPX4PVR (pivot_root) example.....	1368
BPX4PWD (__passwd, __passwd__applid) example.....	1368
BPX4QCT (msgctl) example.....	1368
BPX4QDB (querydub) example.....	1369
BPX4QGT (msgget) example.....	1369
BPX4QRC (msgrcv) example.....	1369
BPX4QSE (quiesce) example.....	1370
BPX4QSN (msgsnd) example.....	1370
BPX4RCV (recv) example.....	1370
BPX4RDD (readdir) example.....	1371
BPX4RDL (readlink) example.....	1371
BPX4RDV (readv) example.....	1371
BPX4RDX (read extlink) example.....	1372
BPX4RD2 (readdir2) example.....	1372
BPX4RED (read) example.....	1372
BPX4REN (rename) example.....	1373
BPX4RFM (recvfrom) example.....	1373
BPX4RLA (readlinkat) example.....	1373
BPX4RMD (rmdir) example.....	1374
BPX4RMG (resource) example.....	1374
BPX4RMS (recvmsg) example.....	1374
BPX4RNA (renameat) example.....	1375
BPX4RPH (realpath) example.....	1375
BPX4RW (Pwrite) example.....	1375
BPX4RWD (rewinddir) example.....	1376
BPX4SA2 (__sigactionset) example.....	1376
BPX4SCT (semctl) example.....	1376
BPX4SDD (setdubdefault) example.....	1377
BPX4SEC (__login, __login__applid, __certificate) example.....	1377
BPX4SEG (setegid) example.....	1378
BPX4SEL (select) example.....	1378
BPX4SEU (seteuid) example.....	1378
BPX4SF (send_file) example.....	1379
BPX4SGE (setgrent) example.....	1379
BPX4SGI (setgid) example.....	1379
BPX4SGQ (sigqueue) example.....	1379
BPX4SGR (setgroups) example.....	1380
BPX4SGT (semget) example.....	1380
BPX4SHT (shutdown) example.....	1380
BPX4SIA (sigaction) example.....	1381
BPX4SIN (server_init) example.....	1381
BPX4SIP (sigpending) example.....	1381
BPX4SLK (shmlock) example.....	1382
BPX4SLP (sleep) example.....	1382
BPX4SMF (smf_record) example.....	1382
BPX4SMS (sendmsg) example.....	1383
BPX4SND (send) example.....	1383

BPX4SNS (setns) example.....	1384
BPX4SOC (socket or socketpair) example.....	1384
BPX4SOP (semop) example.....	1384
BPX4SPB (queue_interrupt) example.....	1385
BPX4SPE (setpwent) example.....	1385
BPX4SPG (setpgid) example.....	1385
BPX4SPM (sigprocmask) example.....	1385
BPX4SPN (spawn) example.....	1386
BPX4SPR (setpeer) example.....	1387
BPX4SPW (server_pwu) example.....	1387
BPX4SPY (setpriority) example.....	1387
BPX4SRG (setregid) example.....	1388
BPX4SRL (setrlimit) example.....	1388
BPX4SRU (setreuid) example.....	1388
BPX4SRX (srx_np) example.....	1389
BPX4SSI (setsid) example.....	1389
BPX4SSU (sigsuspend) example.....	1389
BPX4STA (stat) example.....	1389
BPX4STE (set_timer_event) example.....	1390
BPX4STF (w_statvfs) example.....	1390
BPX4STL (set_thread_limits) example.....	1390
BPX4STO (sendto) example.....	1391
BPX4STR (setitimer) example.....	1391
BPX4STV (statvfs) example.....	1391
BPX4STW (sigtimedwait) example.....	1392
BPX4SUI (setuid) example.....	1392
BPX4SWT (sigwait) example.....	1392
BPX4SYA (symlinkat) example.....	1392
BPX4SYC (sysconf) example.....	1393
BPX4SYF (syncfs) example.....	1393
BPX4SYM (symlink) example.....	1393
BPX4SYN (sync) example.....	1394
BPX4TAF (MVSThreadAffinity) example.....	1394
BPX4TAK (takesocket) example.....	1394
BPX4TDR (tcdrain) example.....	1394
BPX4TFH (tcflush) example.....	1395
BPX4TFW (tcflow) example.....	1395
BPX4TGA (tcgetattr) example .....	1395
BPX4TGC (tcgetcp) example.....	1395
BPX4TGP (tcgetpgrp) example.....	1396
BPX4TGS (tcgetsid) example.....	1396
BPX4TIM (times) example.....	1396
BPX4TLS (pthread_security_np) example.....	1396
BPX4TRU (truncate) example.....	1397
BPX4TSA (tcsetattr) example.....	1397
BPX4TSB (tcsendbreak) example.....	1397
BPX4TSC (tcsetcp) example.....	1397
BPX4TSP (tcsetpgrp) example.....	1398
BPX4TST (tcsettables) example.....	1398
BPX4TYN (ttyname) example.....	1399
BPX4UMK (umask) example.....	1399
BPX4ULA (unlinkat) example.....	1399
BPX4UMT (umount) example.....	1399
BPX4UNA (uname) example.....	1400
BPX4UNL (unlink) example.....	1400
BPX4UPT (unlockpt) example .....	1400
BPX4UQS (unquiesce) example.....	1400
BPX4UNS (unshare) example.....	1401

BPX4UTI (utime) example .....	1401
BPX4WAT (wait) example.....	1401
BPX4WLM (__WLM) example.....	1402
BPX4WRT (write) example.....	1402
BPX4WRV (writev) example.....	1402
BPX4WTE (wait extension) example.....	1403
Reentrant return linkage.....	1403
<b>Appendix F. Examples of nonreentrant entry linkage.....</b>	<b>1407</b>
Example of nonreentrant entry linkage—AMODE 31.....	1407
Example of nonreentrant entry linkage—AMODE 64.....	1409
<b>Appendix G. The relationship of z/OS UNIX signals to callable services.....</b>	<b>1413</b>
High-level-language signal interfaces.....	1413
How high-level languages use signals.....	1414
Signal setup when linking to callable services.....	1414
ESPIE or ESTAE and the SIGILL, SIGFPE, and SIGSEGV signals.....	1415
When signals are and are not supported.....	1415
Signal delivery keys.....	1416
Delayed signal delivery.....	1416
When signals cannot be delivered.....	1417
Signals and multiple tasks created by ATTACH.....	1417
Signals and multiple tasks created by pthread_create.....	1417
Signal defaults.....	1418
<b>Appendix H. Using threads with callable services.....</b>	<b>1421</b>
Creating threads.....	1421
The pthread_create task initialization routine.....	1421
Terminating pthreads.....	1422
Heavyweight thread (HWT).....	1422
Mediumweight thread (MWT).....	1422
Terminating multiple pthreads and tasks.....	1422
Pthread termination scenarios.....	1423
<b>Appendix I. Optimizing performance using process- and thread-level information.....</b>	<b>1431</b>
Optimization processing for BPX1PSI, BPX4PSI (pthread_setintr).....	1431
Optimization processing for BPX1PST, BPX4PST (pthread_setintrtype).....	1432
Optimization processing for BPX1SPM, BPX4SPM (sigprocmask).....	1432
Optimization processing for BPX1GPI, BPX4GPI (getpid).....	1433
<b>Appendix J. Process start and end exits for z/OS UNIX.....</b>	<b>1435</b>
Exit environment.....	1436
Errno and errnoJrs.....	1436
Restrictions.....	1436
Usage notes.....	1437
<b>Appendix K. Dynamic syscall exits for z/OS UNIX.....</b>	<b>1439</b>
Enabling syscalls for exits.....	1439
Defining exit points for syscalls.....	1439
<b>Appendix L. Callable service numbers.....</b>	<b>1445</b>
<b>Appendix M. Accessibility.....</b>	<b>1451</b>
<b>Notices.....</b>	<b>1453</b>

Terms and conditions for product documentation.....	1454
IBM Online Privacy Statement.....	1455
Policy for unsupported hardware.....	1455
Minimum supported hardware.....	1455
Acknowledgments.....	1456
Trademarks.....	1456
<b>Index.....</b>	<b>1457</b>

---

# Figures

1. Call parameter list..... 4

2. setuid() parameters AMODE 31 and AMODE 64..... 9

3. shmget() parameters AMODE 31 and AMODE 64..... 10

4. loadhfs() parameters AMODE 31 and AMODE 64..... 11

5. readv iov structure AMODE 31 and AMODE 64..... 12

6. Program flow of mvssigsetup and sigaction with signal interface routine (SIR)..... 1414





---

# Tables

1. Callable services with no BPX4xxx counterparts.....	8
2. Correspondence between BPXYATT and BPXYSTAT.....	86
3. Attribute fields that can be modified by chattr.....	87
4. Attribute fields that can be modified by fchattr.....	194
5. Correspondence between BPXYATT and BPXYSTAT.....	198
6. Dcor_Request options.....	349
7. PTRACE service options for the Dcor_Request parameter.....	350
8. BPX1SEC/BPX4SEC return values for certificate registration or deregistration with initACEE return code 8.....	368
9. BPX1SEC/BPX4SEC parameter usage based on function requested.....	369
10. Attribute fields that can be modified by lchattr.....	375
11. Correspondence between BPXYATT and BPXYSTAT.....	379
12. Number of input and output arguments and the scope of each defined Function_code.....	504
13. RACF return and reason codes.....	533
14. Poecb control block: Socket scope.....	562
15. Poecb control block: Process scope.....	562
16. Poecb control block: Thread scope.....	562
17. RACROUTE parameters for POE data.....	563
18. POE data propagation for z/OS UNIX services.....	564
19. Resources that can be limited by prlimit.....	575
20. RACF reason and return codes for EMVSSAFEXTRERR.....	606
21. Return codes for ptrace .....	622
22. RACF return and reason codes for the ptrace authority check service .....	626

23. Constant options for the ptrace request parameter.....	626
24. Parameter attributes for request options.....	628
25. Return values and return codes for request options.....	631
26. Corresponding ptrace event and status that is reported to the debugger.....	635
27. Offsets and associate information that are defined in BPXYPTRC.....	642
28. Calling parameters and commands.....	716
29. Options and requirements for set_dub_default.....	754
30. Resources that can be limited by setrlimit.....	790
31. Allowable thread limits for MaxThreadTasks.....	797
32. Allowable thread limits for MaxThread.....	797
33. Correspondence between BPXYATT and BPXYSTAT.....	1008
34. Authorization requirements for __wlm functions.....	1023
35. Correspondence between BPXYATT and BPXYSTAT.....	1028
36. System control offsets to callable services.....	1045
37. Support of signal calls.....	1415
38. Using exit or _exit when the thread is not the IPT.....	1423
39. Using exit or _exit when the thread is the IPT.....	1424
40. Using pthread_exit_and_get when the thread is not the IPT and not the last thread.....	1425
41. Using pthread_cancel when the thread is not the last thread and is canceled.....	1426
42. Using pthread_exit_and_get when the thread is the IPT and not the last thread.....	1426
43. Using pthread_exit_and_get when the thread is not the IPT and is the last thread.....	1427
44. Using pthread_exit_and_get when the IPT is the last thread.....	1428
45. Optimization processing for BPX1PSI, BPX4PSI (pthread_setintr).....	1431
46. Optimization processing for BPX1PST, BPX4PST (pthread_setintrtype).....	1432
47. Optimization processing for BPX1SPM, BPX4SPM (sigprocmask).....	1433

48. Environment at entry to exit routines.....	1440
49. Registers at entry to exit routines.....	1440
50. Registers at return from exit routines.....	1440
51. Values of nonzero return codes.....	1441



## About this document

---

This document describes the features and usage requirements for the z/OS UNIX System Services (z/OS UNIX) callable services. These services are interfaces between the z/OS operating system and standard (POSIX or Single UNIX Specification) programming functions that require operating system services. For example, programmers creating runtime library programs use these services. This book also describes callable services that are not related to the standard interfaces.

System programmers coding programs in assembler can use these callable services to obtain the z/OS UNIX services they need. This document contains detailed information—such as the function, requirements, syntax, linkage information, parameters, and usage information—that is needed to use the services. In the appendixes you will find information about:

- System control offsets to callable services
- Mapping macros
- Callable service examples
- The relationship of signals to callable services
- Using threads with callable services
- Optimizing performance using process- and thread-level information
- Callable services available to SRB mode routines
- z/OS UNIX process start/end exits
- Accessibility features
- Notices
- An index

## Who should use this document

---

This document is for assembler programmers who want to use z/OS UNIX System Services.

## z/OS information

---

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

## Discussion list

Customers and IBM participants also discuss z/OS UNIX on the mvs-oe discussion list. This list is not operated or sponsored by IBM.

To subscribe to the mvs-oe discussion list, send a note to:

```
listserv@vm.marist.edu
```

Include the following line in the body of the note, substituting your given name and family name as indicated:

```
subscribe mvs-oe given_name family_name
```

After you have been subscribed, you will receive further instructions on how to use the mailing list.



## How to provide feedback to IBM

---

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).





## Summary of changes

---

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

**Note:** IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) ([www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy)).

## Summary of changes for z/OS 3.2

---

The following content is new, changed, or no longer included in z/OS 3.2.

### New

The following content is new.

#### September 2025 release

- None.

### Changed

The following content is changed.

#### September 2025 release

- Fields are added in [“BPXZOAPB — z/OS UNIX address space per-process extension”](#) on page 1179.
- [“Exit routine environment”](#) on page 1440 is updated.

### Deleted

The following content is deleted.

#### September 2025 release

- None.

## Summary of changes for z/OS 3.1

---

The following content is new, changed, or no longer included in z/OS 3.1.

### New

The following content is new.

#### December 2024 refresh

- In support of [OA66872: Z/OS UNIX LONG-RUNNING DOC APAR 202408](#) ([www.ibm.com/support/pages/apar/OA66872](http://www.ibm.com/support/pages/apar/OA66872)), updates are made to [“Usage notes for \\_\\_getthent”](#) on page 327. (APAR OA66872, which applies to z/OS 2.4 and later)

#### August 2024 refresh

- In support of [OA66242: Z/OS UNIX LONG-RUNNING DOC APAR 202403](#) ([www.ibm.com/support/pages/apar/OA66242](http://www.ibm.com/support/pages/apar/OA66242)), updates are made to the fork callable service. See [“MVS-related information”](#) on page 232. (APAR OA66242, which applies to z/OS 2.4 and later)

## June 2024 refresh

- In support of [OA66410: NEW FUNCTION - USS Support for new APIs to Satisfy Compiler Requirement \(www.ibm.com/support/pages/apar/OA66410\)](#), updates are made to Appendix L, “Callable service numbers,” on page 1445. (APAR OA66410, which also applies to z/OS 2.5)

## March 2024 refresh

- In support of [OA61972: NEW FUNCTION - Z/OS UNIX SUPPORT FOR NAMESPACES \(www.ibm.com/support/pages/apar/OA61972\)](#), new services are added.
  - [“accept4 \(BPX1AC4, BPX4AC4\) — Accept a connection request from a client socket and set flags ” on page 16.](#)
  - [“clone \(BPX1CLN, BPX4CLN\) — Create a child process” on page 114.](#)
  - [“epoll\\_ctl \(BPX1EPL, BPX4EPL\) — Control interface for epoll file descriptor” on page 152.](#)
  - [“epoll\\_create \(BPX1EPC, BPX4EPC\) — Open an epoll file descriptor ” on page 150.](#)
  - [“epoll\\_wait \(BPX1EPW, BPX4EPW\) — Wait for an I/O event on an epoll file descriptor” on page 157.](#)
  - [“eventfd \(BPX1EFD, BPX4EFD\) — Create a file descriptor for event notification ” on page 160.](#)
  - [“inotify\\_add\\_watch \(BPX1IAW, BPX4IAW\) — Add or modify a watch to an initialized inotify instance” on page 338.](#)
  - [“inotify\\_init \(BPX1INI, BPX4INI\) — Initialize an inotify instance” on page 341.](#)
  - [“inotify\\_rm\\_watch \(BPX1IRW, BPX4IRW\) — Remove existing watches from inotify instances” on page 344 is added.](#)
  - [“pipe2 \(BPX1PI2, BPX4PI2\) — Create a new pipe instance ” on page 553.](#)
  - [“prctl \(BPX1PRC, BPX4PRC\) — Managing processes and threads ” on page 568.](#)
  - [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ” on page 574.](#)
  - [“setns \(BPX1SNS, BPX4SNS\) — Reassociate thread with a namespace” on page 771.](#)
  - [“unshare \(BPX1UNS, BPX4UNS\) — Run program in a new namespace” on page 977.](#)

(APAR OA61972 applies to z/OS 2.5 and later.)

## January 2023 refresh

- In support of [OA64356: NEW FUNCTION - ADD NEW \\_\\_AUTHENTICATE\(\) SYSCALL TO SUPPORT IDENTITY TOKENS \(www.ibm.com/support/pages/apar/OA64356\)](#), various updates are added. See [“\\_\\_authenticate \(BPX1AUT, BPX4AUT\) — Authenticate the specified user's credentials” on page 69.](#) [“List of offsets” on page 1045](#) and [“BPXYCONS — Constants used by services” on page 1062](#) are also updated. (APAR OA64356, which applies to z/OS 2.4 and later)

## November 2023 refresh

- In support of [PH57159: REPLACING SYSTEM CALL \(www.ibm.com/support/pages/apar/PH57159\)](#), updates are added to [“mmap \(BPX1MMP, BPX4MMP\) — Map pages of memory ” on page 435.](#) [“BPXYCONS — Constants used by services” on page 1062](#) is also updated.

## September 2023 release

- With the getrandom service, you can create a new getrandom instance that returns random data. See [“getrandom \(BPX1GRD, BPX4GRD\) — Obtain a series of random bytes” on page 310.](#)
- You can use extended attributes to control the properties of files and directories. These callable services are updated:
  - [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory ” on page 82](#)
  - [“fchattr \(BPX1FCR, BPX4FCR\) — Change the attributes of a file or directory by descriptor” on page 190](#)

- [“lchattr \(BPX1LCR, BPX4LCR\) — Change the attributes of a file or directory or symbolic link” on page 371](#)
- [“w\\_ioctl \(BPX1IOC, BPX4IOC\) — Control I/O ” on page 1003](#)
- [“w\\_piocctl \(BPX1PIO, BPX4PIO\) — Path name I/O control” on page 1025](#)

These mapping macros are also updated:

- [“BPXYATT — Map file attributes for chattr and fchattr” on page 1057](#)
- [“BPXYIOCC — Ioctl command definitions” on page 1085](#)
- Use the chattrat callable service to change the attributes of files and directories. [“chattrat \(BPX1CRA, BPX4CRA\) — Change the attributes of a file or directory relative to a directory file descriptor” on page 91.](#)
- Use the flock callable service to apply or remove an advisory lock on the open file that is specified by a file descriptor, fd. See [“flock \(BPX1FLK, BPX4FLK\) — Apply or remove an advisory lock on an open file” on page 225.](#)
- Use the syncfs service to synchronize the file system file that the open file descriptor refers to. See [“syncfs \(BPX1SYF, BPX4SYF\) — Synchronizes the file system file referred to by the open file descriptor ” on page 912.](#) The F\_DUPFD3 action and the F\_DUP3 control block are added to BPX1FCT. See [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216.](#)
- Use the prctl callable service to set or query the attributes of various processes or threads, depending on the subfunction that is requested. See [“prctl \(BPX1PRC, BPX4PRC\) — Managing processes and threads ” on page 568.](#)
- Use the prlimit callable service, you can get and set process resource limits. See [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ” on page 574.](#)
- You can use the clone, setns, and unshare callable services to manage namespaces. New callable services are added:
  - [“clone \(BPX1CLN, BPX4CLN\) — Create a child process” on page 114.](#) [“Defining exit points for syscalls” on page 1439](#) is also updated.
  - [“setns \(BPX1SNS, BPX4SNS\) — Reassociate thread with a namespace” on page 771](#)
  - [“unshare \(BPX1UNS, BPX4UNS\) — Run program in a new namespace” on page 977](#)

Other updates are made.

- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)
- [“Defining exit points for syscalls” on page 1439](#)
- When you are working with the epoll file descriptors, you can use the epoll family of syscalls.
  - [“epoll\\_create \(BPX1EPC, BPX4EPC\) — Open an epoll file descriptor ” on page 150](#)
  - [“epoll\\_ctl \(BPX1EPL, BPX4EPL\) — Control interface for epoll file descriptor” on page 152](#)
  - [“epoll\\_wait \(BPX1EPW, BPX4EPW\) — Wait for an I/O event on an epoll file descriptor” on page 157](#)
- The eventfd callable service monitors and updates notification events. See [“eventfd \(BPX1EFD, BPX4EFD\) — Create a file descriptor for event notification ” on page 160.](#)
- The fchownat service to change ownership of a file relative to a directory file descriptor. See [“fchownat \(BPX1COA, BPX4COA\) — Update the file ownership of a file or directory relative to a directory file descriptor ” on page 213.](#) You can also use the symlinkat service to create a symbolic link relative to a directory file descriptor. See [“symlinkat \(BPX1SYA, BPX4SYA\) — Create a symbolic link relative to a directory file descriptor ” on page 905.](#)
- The faccessat service. See [“faccessat \(BPX1FAA, BPX4FAA\) — Checks user's permissions of a file relative to a directory file descriptor” on page 186.](#)
- The fchmodat service modifies the permission bits that control access. You can also allow a starting directory other than the current directory to be used when a relative path name is specified. See

[“fchmodat \(BPX1CMA, BPX4CMA\) — Change the mode of a file or directory relative to a directory file descriptor” on page 206.](#)

- The `fstatat` service to obtain the status of a file. See [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor” on page 241.](#)
- You can use the `inotify` family of syscalls to monitor changes in file systems and send notifications.
  - [“inotify\\_add\\_watch \(BPX1IAW, BPX4IAW\) — Add or modify a watch to an initialized inotify instance” on page 338](#)
  - [“inotify\\_init \(BPX1INI, BPX4INI\) — Initialize an inotify instance” on page 341](#)
  - [“inotify\\_rm\\_watch \(BPX1IRW, BPX4IRW\) — Remove existing watches from inotify instances” on page 344](#)
- The `linkat` service creates a link to an existing file by using a directory file descriptor as the starting location for the path searches for the link and the file. See [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor” on page 387.](#)
- The `mkdirat` service creates a directory relative to a directory file descriptor. See [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor” on page 423.](#)
- The `mknodat` service makes a directory, a FIFO, a character-special, or a regular file relative to a directory file descriptor. See [“mknodat \(BPX1MNA, BPX4MNA\) — Make a directory, a FIFO, a character special, or a regular file relative to a directory file descriptor” on page 431.](#)
- Use the `openat` and `openat2` callable services to control how path names are resolved during the open. See [“openat/openat2 \(BPX1OAT, BPX4OAT\) — Open a file with path name parameters relative to a directory file descriptor” on page 518.](#)
- Use the `pivot_root` service to change the root mount in the mount namespace of the calling process. See [“pivot\\_root \(BPX1PVR, BPX4PVR\) — Change the root mount” on page 556.](#)
- The `readlinkat` service reads the value of a symbolic link relative to a directory file descriptor. See [“readlinkat \(BPX1RLA, BPX4RLA\) — Read the value of a symbolic link relative to a directory file descriptor” on page 669.](#)
- Use the `wait4()` function that is provided by the `wait-extension` callable service to obtain status information for its child processes. See [“wait-extension \(BPX1WTE, BPX4WTE\) — Obtain status information for children” on page 987.](#)
- Communication between processes is simplified with these new callable services.
  - [“accept4 \(BPX1AC4, BPX4AC4\) — Accept a connection request from a client socket and set flags” on page 16](#)
  - [“pipe2 \(BPX1PI2, BPX4PI2\) — Create a new pipe instance” on page 553](#)

The following callable service is updated with new flags:

- [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets” on page 865](#)
- The `BPXYOPNF` macro is updated with new flags to set and retrieve various information about OPEN processing. See [“BPXYOPNF — Map flag values for open” on page 1113.](#)
- The `renameat` service renames a file by using the `at` directory relative to a directory file descriptor. See [“renameat \(BPX1RNA, BPX4RNA\) — Rename a file using the at directory relative to a directory file descriptor” on page 694.](#)
- The `unlinkat` service removes a directory or directory entry relative to a directory file descriptor. See [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969.](#)

## Changed

The following content is changed.

### June 2025 refresh

- Minor updates are made.

## May 2025 refresh

- A usage note is added to [“poll \(BPX1POL, BPX4POL\) — Monitor activity on file descriptors and message queues”](#) on page 564.

## March 2025 refresh

- Additional information is added about the flags that can be specified on an open call. See [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511.
- In support of [APAR OA66002 \(www.ibm.com/support/pages/apar/OA66002\)](#), updates are made to [“Summary of option behavior and requirements”](#) on page 754 and [“Usage notes for set\\_dub\\_default”](#) on page 756. A new usage note is also added to [“pthread\\_quiesce \(BPX1PTQ, BPX4PTQ\) — Quiesce threads in a process”](#) on page 598. (APAR OA66002, which also applies to z/OS 3.1 and z/OS 2.5)
- Information about AT-TLS is added to the EWOULDBLOCK return code for the following services:
  - [“read \(BPX1RED, BPX4RED\) — Read from a file or socket”](#) on page 652
  - [“readv \(BPX1RDV, BPX4RDV\) — Read data and store it in a set of buffers”](#) on page 673
  - [“recv \(BPX1RCV, BPX4RCV\) — Receive data on a socket and store it in a buffer”](#) on page 680
  - [“recvfrom \(BPX1RFM, BPX4RFM\) — Receive data from a socket and store it in a buffer”](#) on page 683
  - [“recvmsg \(BPX2RMS, BPX4RMS\) — Receive messages on a socket and store them in message buffers”](#) on page 687

## February 2025 refresh

- The usage notes in [“eventfd \(BPX1EFD, BPX4EFD\) — Create a file descriptor for event notification”](#) on page 160 are updated.

## July 2024 refresh

- [“BPXYOPNF — Map flag values for open”](#) on page 1113 is updated for clarity.
- [“BPXYPGTH — Map the \\_\\_getthent input/output structure”](#) on page 1119 is updated.

## May 2024 refresh

- The description of the O\_DIRECT flag is updated. See [“pipe2 \(BPX1PI2, BPX4PI2\) — Create a new pipe instance”](#) on page 553.

## March 2024 refresh

- In support of [OA65077: Z/OS UNIX LONG-RUNNING DOC APAR 202306 \(www.ibm.com/support/pages/apar/OA65077\)](#), usage notes for new environment variables are added. See [“exec \(BPX1EXC, BPX4EXC\) — Run a program”](#) on page 163. (APAR OA62281, which applies to z/OS 2.5 and later)
- In support of [OA65077: Z/OS UNIX LONG-RUNNING DOC APAR 202306 \(www.ibm.com/support/pages/apar/OA65077\)](#), [“read \(BPX1RED, BPX4RED\) — Read from a file or socket”](#) on page 652 and [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket”](#) on page 1032 are updated.  
(APAR OA65077, which applies to z/OS 2.5 and later)

## February 2024 refresh

- In support of [OA62734: NEW FUNCTION - Z/OS UNIX SUPPORT FOR MOUNT NAMESPACES \(www.ibm.com/support/pages/apar/OA62734\)](#), updates are made for [“\\_\\_mount \(BPX2MNT, BPX4MNT\) — Make a file system available”](#) on page 447 and [“BPXYMNTTE — Map response and element structure of w\\_getmntent”](#) on page 1104. (APAR OA62734, which applies to z/OS 2.5 and later)

## November 2023 refresh

- In support of [OA62733: NEW FUNCTION - SUPPORT NEW CONTAINER RELATED SYSCALLS \(www.ibm.com/support/pages/apar/OA62733\)](#), updates are made to [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216. (APAR OA62733, z/OS 2.5 only)

**October 2023 refresh**

Various updates are made.

**September 2023 release**

- A usage note is added about Unicode Services in [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#) and [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#).

**Deleted**

The following content is deleted.

**September 2023 release**

- None.

# Chapter 1. Invocation details for callable services

As an interface between the z/OS operating system and the functions specified in the Single UNIX Specification and earlier standards, z/OS UNIX System Services (z/OS UNIX) provides access to assembler callable services (syscalls). The z/OS UNIX callable services have a standard set of syntax and linkage requirements, as well as parameter specification details necessary for successful invocation.

## Connecting to and disconnecting from z/OS UNIX

To connect to the kernel for z/OS UNIX System Services, you make an address space known to it. This process is called *dubbing*. After it has been dubbed, the address space is considered to be a *process*. Address spaces that are created by fork are automatically dubbed when they are created. Other address spaces become dubbed if they invoke a z/OS UNIX service. Dubbing also applies to MVS tasks. A dubbed task is considered a *thread*. Tasks that are created by pthread\_create are automatically dubbed threads; other tasks are dubbed if they invoke a z/OS UNIX service.

*Undub* is the inverse of *dub*. Normally, a task (dubbed a thread) is undubbed when it ends. An address space (dubbed a process) is undubbed when the last thread ends.

If, when a thread or process is being dubbed, the calling task has a task-level ACEE that does not have a USP connected to it, an INITUSP is done against the task-level ACEE. This causes z/OS UNIX security information to be associated with the task-level ACEE.

Every address space that is dubbed is required to have an address space level security environment with a valid OMVS segment. In a multi-user address space, if a task is dubbed and it has a security environment different from the address space, then the user ID that is identified with the task must also have a valid OMVS segment.

## Syntax conventions for the callable services

A callable service is a programming interface that uses the CALL macro to access system services. To code a callable service, code the CALL macro followed by the name of the callable service and a parameter list. A syntax diagram for a callable service follows.

```
CALL Service_name, (Parm_1,
                   Parm_2,
                   .
                   .
                   Return_value,
                   Return_code,
                   Reason_code)
```

This format does not show the assembler column conventions (columns 1, 10, 16, and 72) or parameter list options (VL and MF). The exact syntax is shown in the examples in [Appendix D, “Callable services examples—AMODE 31,”](#) on page 1219.

When you code a callable service you must:

- Code all the parameters in the parameter list, because parameters are positional in a callable service interface. The function of each parameter is determined by its position with respect to the other parameters in the list. Omitting a parameter, therefore, assigns the omitted parameter's function to the next parameter in the list.
- Place values explicitly into all supplied parameters, because callable services do not set defaults.

## CALL

CALL is the assembler macro that transfers control and passes a parameter list.



### Service\_name

For AMODE 31 callers, the name that the assembler understands is the name of a module in the form BPX1xxx, where xxx is a three-character symbol unique to the service. (In a few cases, where both standard and nonstandard versions of a service exist, the standard version of the service is in the form BPX2xxx.) AMODE 64 callers use the name of a module in the form BPX4xxx. (See [“Using callable services in a 64-bit environment”](#) on page 8.)

Modules are invoked in one of the following ways:

- A program can load a module, and then branch to the address where it was loaded.
- When you link-edit a program, you can link to the linkage stub. The program can issue a call.

The linkage stubs are contained in SYS1.CSSLIB. You can specify SYS1.CSSLIB in the //SYSLIB statement of the JCL that is used to invoke the linkage editor. This causes the addresses of all required linkage-assist routines to be automatically resolved, and saves you the trouble of having to specify individual linkage-assist routines in INCLUDE statements.

For BPX4xxx stubs, you need 64-bit binder support to do the link-edit. See [“Using callable services in a 64-bit environment”](#) on page 8.

- You can include in the code the system control offset to the callable service. See Appendix A, [“System control offsets to callable services,”](#) on page 1045 for information on how to use this linkage.

For information about using callable services in AMODE 64, see [“Using callable services in a 64-bit environment”](#) on page 8.

### Parm parameters

Parm\_1, Parm\_2, and so on are placeholders for variables that may be part of a service's syntax.

### Return\_value

The Return\_value parameter is common to many callable services, and indicates the success or failure of the service. If the callable service fails, it returns a -1 in the Return\_value. For most successful calls to z/OS UNIX services, the return value is set to 0. However, some services, such as [“getgrgid \(BPX1GGI, BPX4GGI\) — Access the group database by ID”](#) on page 268 and [“getgrnam \(BPX1GGN, BPX4GGN\) — Access the group database by name”](#) on page 271, return zeros instead of -1 when the service fails.

Some callable services, such as [“fork \(BPX1FRK, BPX4FRK\) — Create a new process”](#) on page 227, return a positive return value to indicate success. Other services, such as [“\\_exit \(BPX1EXI, BPX4EXI\) — End a process and bypass the cleanup”](#) on page 180, are unique in that they do not return when successful.

Some services do not have a return value, because under normal conditions they do not fail. System failures, however, may cause these services to fail, in which case, the process that issues the call ends abnormally. See [“getegid \(BPX1GEG, BPX4GEG\) — Get the effective group ID”](#) on page 263 for an example.

### Return\_code

The Return\_code parameter is referred to as the *errno* in the POSIX and X/Open C interfaces. The Return\_code is returned only if the service fails.

In the callable service descriptions, some of the possible return codes are listed for services that have return codes. The return codes are described in each service if they help to describe its function.

For each return code, any reason code that may accompany it is identified.

The return codes and their descriptions can be found in [Return codes \(errno\) in z/OS UNIX System Services Messages and Codes](#).

Some Return\_code values may occur for any callable service: the return codes that are unique to z/OS UNIX. They are not always listed under each callable service.



The following five return codes can occur with any callable service, and are not listed with each service because the failure may occur before the syscall gets control:

Return_code	Explanation
EFAULT	An address is incorrect, usually because it is a zero pointer, an uninitialized pointer, or a pointer to read-only storage (for example, a program constant of zero) for a parameter that is (or could be, in a different context) an output parameter.
EMVSINITIAL	A process initialization error has occurred.
EMVSERR	An environmental or internal error has occurred.
EMVSPARM	Bad parameters were passed to the service.
ENOMEM	Not enough space is available to fill the request.

## Reason\_code

The Reason\_code parameter usually accompanies the Return\_code value when the callable service fails, and further defines the return code. Reason codes do not have an equivalent in the POSIX or X/Open standards.

The reason codes and their descriptions can be found in [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*. Reason codes are listed by name and numerically by value. The value is the lower half of the reason code.

## Determining the callable service release level

New callable services may be added with each new z/OS UNIX release. Depending on the operating environment, the caller may have to determine the release level of z/OS UNIX before a new callable service can be issued.

The release information is indicated in the CVT feature flags. For z/OS V1R1, the feature flag is:

```
CVTJ7713 EQU    X'20'      JBB7713
```

## Linkage conventions for the callable services

Callers must use the following linkage conventions for all z/OS UNIX callable services:

- Register 1 is set up by the CALL macro. In 31-bit mode, it contains the address of a parameter list, which is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit. In 64-bit mode, register 1 is 8 bytes long, and contains a 64-bit address that points to a list of 64-bit addresses. See [“Using callable services in a 64-bit environment”](#) on page 8.
- Register 14 is set up by the CALL macro; it contains the return address.
- Register 15 is set up by the CALL macro; it contains the entry point address of the service stub that is being called.

On return from a callable service, general and access registers 2 through 13 are restored. General and access registers 0, 1, 14, and 15 are not restored.

The caller does not have to provide the address of a save area in register 13.

The caller must be running with 31-bit or 64-bit addressing (AMODE=31 or AMODE=64), because the linkage code uses control blocks that reside above the 16-MB line.

For more information about linkage, see [Linkage conventions](#) in *z/OS MVS Programming: Assembler Services Guide*.

## Parameter descriptions for the callable services

All the parameters of the callable services are required positional parameters. When you specify a call, you must specify all the parameters in the order listed.

**Note:** Some parameters do not require values, and allow you to substitute zeros for the parameter. The descriptions of the parameters identify those that can be replaced by zeros, and when to do so.

In the descriptions of the calls, each parameter is described as *supplied* or *returned*:

- *Supplied* means that you supply a value for the parameter in the call.
- *Returned* means that the service returns a value in the named parameter when the call is finished (for example, *Return\_code*).
- Some parameters are both supplied and returned.

Each parameter is also described in terms of its *data type* and *length*:

- The data type is one of the following: integer, address, character string, or structure.
- The length depends on the data type of the parameter:
  - For an address item, the length is a fullword (for AMODE 31 callers) or a doubleword (for AMODE 64 callers).
  - For an integer item, the length indicates the size of the field in bytes or fullwords: bytes are 1, 2, 3, 4, or 8.
  - For a character string parameter, the length indicates the number of characters that can be contained in a character-type parameter.
  - For a structure parameter, the length indicates the size of the structure in bytes or fullwords, or refers to a label in the structure's mapping macro that defines the length.

## Call parameter lists

Every callable service is called with a parameter list. As shown in the following figure, when a service is called:

- Register 1 points to a parameter address list.
- Each field in the parameter address list points to a field containing a parameter.
- The parameter list is the set of those parameters; however they are arranged in storage. For AMODE 31 callers, the last parameter pointer in the list must have the high-order bit set to 1. For AMODE 64 callers, there is no end-of-parameter list indicator; the high-order bit is part of the 64-bit address.

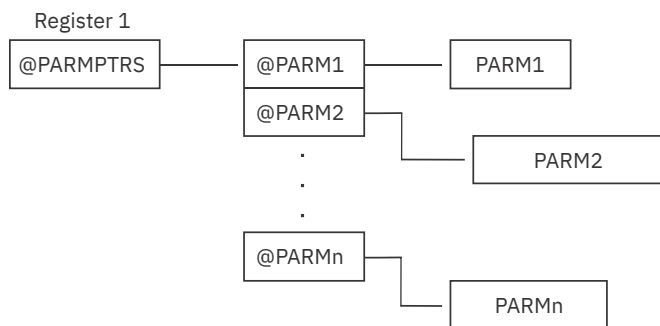


Figure 1. Call parameter list

## Mapping macros

In many callable services, mapping macros map the parameter options. A complete list of the options for each macro is listed in the macro in [“Macros mapping parameter options” on page 1055](#).

Most of the mapping macros can be expanded with or without a DSECT statement. The invocation operand DSECT=YES is the default.

AMODE 64 callers using the 64-bit versions of the macros must issue SYSSTATE AMODE64=YES to identify the addressing mode. See [SYSSTATE - Identify system state in z/OS MVS Programming: Assembler Services Reference IAR-XCT](#) for information about the SYSSTATE macro.

## Examples

An invocation example for each callable service is in [Appendix D, “Callable services examples—AMODE 31,” on page 1219](#). These examples follow the rules of reentrancy. They use DSECT=NO and place the variables in the program's dynamic storage DSECT, which is allocated upon entry. The examples are arranged alphabetically, and have references to the mapping macros they use. The declaration for all local variables used in the examples follows the examples.

## Reentrant coding versus nonreentrant coding

See “BPX1GTH (\_\_getthent) example” on page 1251 for an example of the \_\_getthent service. That service uses reentrant code. Compare this example with an example of nonreentrant code for the same service in “[Example of nonreentrant entry linkage—AMODE 31](#)” on page 1407 and note the following:

- Placement of the standard 18-word register save area.
- Use of program and dynamic storage base registers.
- Different forms of the CALL macro.
- Several variables such as PGPSCONTTYBLEN that are initialized by the assembler in the nonreentrant version. (For more information about the DCS, see “[BPXYPGPS — Map the response structure for w\\_getpsent](#)” on page 1116). They are also initialized at execution time with moves and stores in the reentrant version.

## Environmental restrictions

Callers must be aware of the following restrictions for all z/OS UNIX callable services:

### Functional recovery routines (FRR)

Except for callable services that are supported in service request block (SRB) mode, do not invoke a callable service with a functional recovery routine set because doing so bypasses callable services recovery and can severely damage the system. (If a callable service can run in SRB mode, that is stated in its description.)

### Linkage stack

The use of the system linkage stack with PC or BAKR instructions prevents signals from being delivered.

### Locks

Do not call z/OS UNIX with system locks held. Testing is not done for locks held, and your call might fail.

### Nested callable services

You cannot issue “nested” callable services. That is, if a program running on a request block (RB) issues a z/OS UNIX callable service and is then interrupted by a program running on an interrupt request block (IRB), any additional z/OS UNIX callable services that the IRB attempts to issue are not supported. Additionally, if a z/OS UNIX callable service invokes an exit during the processing of the callable service, invoking z/OS UNIX callable services from the exit program is not supported. In most cases, the nested callable service invocation is detected and flagged as an error. In some cases, however, the nested invocation is not detected and can lead to failure of the original callable service invocation.

### Task structure

When you invoke callable services in task control block (TCB) mode, the calling TCB must be either the initial job step task or a subtask of the initial job structure task. The initial job structure task is the job

structure task that is directly attached by the operating system initiator task to run a user requested program. z/OS UNIX does not support the direct attachment of multiple job structure tasks from the initiator task. The behavior of z/OS UNIX callable services in an environment where multiple job step tasks are attached directly from the initiator task is unpredictable.

## Restrictions in a multiprocess, multiuser environment

---

Programs that change the security environment cannot run in a multiprocess, multiuser environment. A multiprocess, multiuser environment is an environment in which there are multiple z/OS UNIX processes in an address space (enabled by the environment variable `_BPX_SHAREAS=YES`.) Each process has a different MVS identity; that is, it has its own process-level ACEE anchored at the TCB (TCBSenv) level. To prevent a user running under one MVS identity from affecting all the other processes in the address space, or creating a new process with an identity other than the one the user is running under, certain callable services are restricted.

These z/OS UNIX callable services are restricted in a multiprocess, multiuser environment, and will fail with JRMpMuProcess:

- BPX1ATM/BPX4ATM (attach\_execMVS) — ASM only
- BPX1ATX/BPX4ATX (attach\_exec) — ASM only
- BPX1SEG/BPX4SEG (setegid)
- BPX1SGI/BPX4SGI (setgid)
- BPX1SPN/BPX4SPN (spawn family)
- BPX1SRG/BPX4SRG (setregid)

See the descriptions of these callable services for further information about the restrictions.

## Restrictions in a multiprocess environment

---

A *multiprocess address space* is an environment in which there are multiple z/OS UNIX processes in an address space. A multiprocess address space is an environment in which there are multiple z/OS UNIX processes in an address space. Multiple processes are created when:

- A local spawn() request is done (environment variable `_BPX_SHAREAS=YES|MUST` has been set).
- A task is dubbed in an address space where the set\_dub\_default() DUBSETTING has been set to DUBPROCESS or DUBPROCESSDEFER.

All processes in the address space share the same MVS identity. Changing the UID in one process will affect the other processes in the address space. This can cause unpredictable results for other processes who were expecting to run with the original address space identity.

For non-TSO address spaces, the following z/OS UNIX callable services are restricted in a multiprocess address space:

Service	Restriction
BPX1SUI/BPX4SUI (setuid)	Sets the UID.
BPX1SEU/BPX4SEU (seteuid)	Sets the effective UID.
BPX1SRU/BPX4SRU (setreuid)	Sets the effective and real UID.

In a TSO address space, BPX1SUI/BPX4SUI is not allowed. However, BPX1SEU/BPX4SEU and BPX1SRU/BPX4SRU are supported. In a TSO address space, only one locally spawned process is allowed at a given time.

Although a spawned program can change the UID, the program should reset the UID to the original value unless it intentionally wants to change the behavior of future spawned programs. If a program does not reset the UID to the original values, either intentionally or because the program ended abnormally, the address space will run with the new UID values.

## Abend conditions and environments

---

Callers must be aware of the following conditions that can cause an abnormal end:

- When the `_exit` service, BPX1EXI/BPX4EXI, is called in any environment except single task, single RB, and no linkage stack, the system issues an abend EC6. This abend ends the calling task and all of its subtasks. The subtasks receive a 442 abend. If the caller is a thread task created with the `pthread_create` service, the initial pthread creating task abends with a 422 abend code. All subtasks of the initial pthread creating task receive a 442 abend.
- Some POSIX services are defined as always successful, yet the kernel can get program checks or other MVS abends. When these failures occur, the user receives an EC6 abend code.
- There are SLIP traps that recognize z/OS UNIX abends as normal exec service and `_exit` service processing. Dumps are suppressed, and the new tasks for the exec service are created. These SLIP traps are shipped as part of IEASLP00. If your system does not use IEASLP00 as provided by z/OS, you will need to copy the SLIP commands for EC6 and 422 abends into their SLIP command parmlib member. Otherwise, your system will generate an excessive number of dumps.
- Condition codes (cc) seen by the next step in a multistep job cause an abnormal end:
  - Case 1:
    1. The step invokes the C main program.
    2. The C main program invokes the `exit` or `_exit` service, specifying the return code.
    3. The return code surfaces as the step condition code.
  - Case 2:

When you return from the main program, the condition code is in R15 at the time of exit.

- Signals that are not caught often cause a task to end abnormally. z/OS UNIX defines which signals generate dumps. Terminating signals that do not require user dumps have an abend code of EC6 with a reason code 0000FFxx, where xx is the signal number. Parmlib member IEASLP00 has a statement to suppress all dumps that match this profile. Terminating signals that require that a user dump be taken (if requested) have an abend code of EC6 with a reason code 0000FDxx, where xx is the signal number. Parmlib member IEASLP00 has a statement to suppress all SDUMPs that match this profile but that allow user dumps to be taken.
- If a process abends while it is being debugged with `ptrace` by a debugger program such as `dbx`, the debugger may be notified of the abend. The notification occurs if the tested program's recovery calls `ptrace`. This is normally true for C programs, because the C runtime library establishes the necessary recovery environment to call `ptrace`.

## Callable service failures

---

A typical application that receives an unexpected return code from a callable service usually exits. If an application is written to handle unexpected errors, you need to understand the following information:

Services can fail for a number of reasons: bugs in the system, user code that causes failure return codes, or abend conditions. Depending on when the failure occurs in the service path, the requested function may or may not have been performed. For example, if the application provides an address for a file descriptor that does not exist, the open service (BPX1OPN/BPX4OPN) completes the open processing and then fails on the return path when trying to set the file descriptor. If an EFAULT return code is returned, the user may assume that the file was not opened, even though it was.

If the return value parameter is not in valid storage, a service can complete successfully, yet not return normally to the caller. Because the service cannot set the return value, it abends. It is possible for the C runtime library to convert the return value into a SIGABND or SIGSEGV signal, which can be caught and handled by the user signal action defined in `sigaction`. You should be aware that functions that abend in this way may have completed their processing. For example, a call to `sigaction` could modify the state of signal information and then fail on the return to the caller. In this case, the caller should not make any assumptions about the state of the signal environment.

## Authorization

Users authorized to perform special functions are defined as having *appropriate privileges* and are called *superusers*. Users with appropriate privileges are also those with:

- A user ID of zero
- RACF-supported user privileges *trusted* and *privileged*, regardless of their user ID

The ability to change the MVS identity of an address space is reserved for a subset of superusers who control daemons. A daemon is a process that verifies the identity of a user before creating a process to run work on behalf of the user. This approach allows the installation to have superusers whose job is to maintain the file system and user processes, but who do not have the ability to change their user identity. See [Setting up the UNIX-related FACILITY and SURROGAT class profiles in z/OS UNIX System Services Planning](#) for a description of the BPX.DAEMON resource profile in the RACF® FACILITY class and how it is created. This information also describes additional BPX.xxxxxxxx resource profiles in the FACILITY class that are used to provide selective permission to certain restricted functions.

Also, superusers are said to have *daemon authority* if the BPX.DAEMON resource profile is defined and they have access to it. If BPX.DAEMON is not defined, the users have daemon authority if they are a superuser.

Note that aliases can be supplied for user IDs. Callable services that pass or receive user ID parameters may need to use the userid alias table. Its use is described in [USERIDALIASTABLE in z/OS UNIX System Services Planning](#).

**Note:** This information assumes that your operating system contains Resource Access Control Facility (RACF). You could use an equivalent security product that was updated to handle z/OS UNIX security.

## Using callable services in a 64-bit environment

The z/OS UNIX callable services can be called in 31-bit or 64-bit addressing mode (AMODE 31 or AMODE 64). RMODE 64 is not supported. AMODE 31 callers use the BPX1xxx services; AMODE 64 callers use the BPX4xxx services.

Calling programs using 64-bit addressing must be compiled AMODE 64, and clear the upper half of the branch register before branching to the syscall layer. 64-bit binder support is needed to do the link-edit.

An AMODE 64 caller using the 64-bit versions of the macros must issue SYSSTATE AMODE64=YES before calling the service. For more information about the SYSSTATE macro, see [SYSSTATE - Identify system state in z/OS MVS Programming: Assembler Services Reference IAR-XCT](#).

There is a BPX4xxx stub for each BPX1xxx service, except for those services that were replaced in functionality by other services, as shown in [Table 1 on page 8](#).

Table 1. Callable services with no BPX4xxx counterparts	
Callable service	Replaced by
<b>BPX1GPS</b>	BPX1GTH
<b>BPX1TYN</b>	BPX2TYN
<b>BPX1ITY</b>	BPX2TYN
<b>BPX1MNT</b>	BPX2MNT
<b>BPX1RMS</b>	BPX2RMS
<b>BPX1SMS</b>	BPX2SMS

The kernel continues to support the BPX1xxx versions of these syscalls.

Except for the services in [Table 1 on page 8](#), the last three characters of the stub names are the same for the 31-bit and 64-bit stubs.

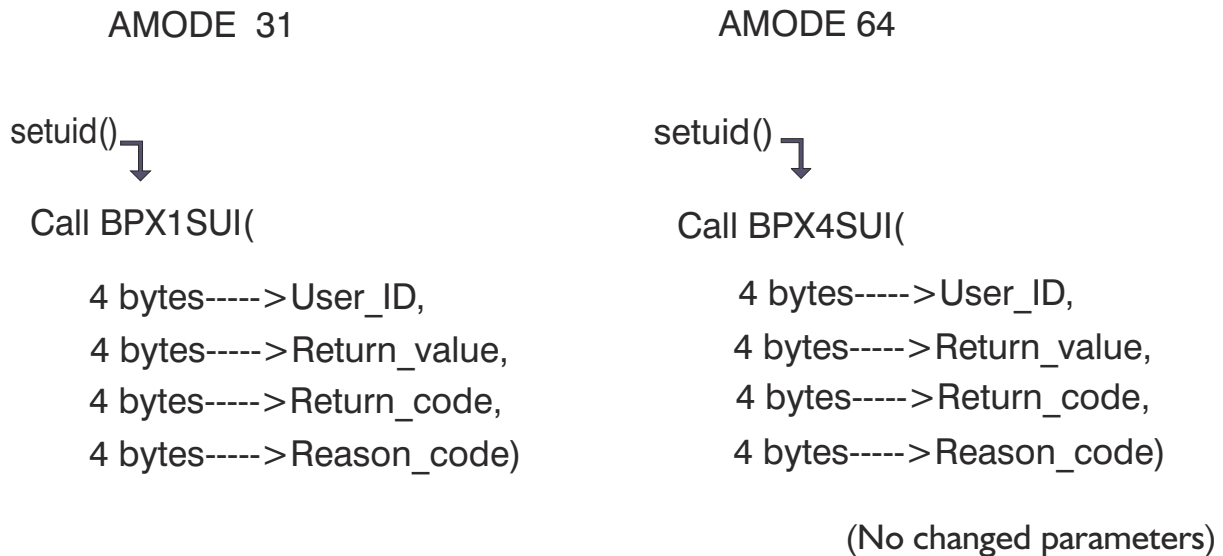
## Call parameter lists

AMODE 31 callers of the BPX1xxx service provide a standard 31-bit parameter list. Register 1 contains a 31-bit address that points to a list of 31-bit addresses, which point to the parameters. The last parameter pointer in the list must have the high-order bit set to 1. All storage is below the bar.

AMODE 64 callers of the BPX4xxx service provide a 64-bit address in register 1 that points to a list of 64-bit addresses, which point to the parameters. There is no end-of-parameter list indicator; the high-order bit is part of the 64-bit address. The parameter list and the parameters may or may not exist above the bar.

## Parameters

For most of the callable services, the parameters for the 31-bit and 64-bit versions are the same; the only difference between the versions is the AMODE of the caller. There is no change in the number or length of the parameters. An example of this type is `setuid()`:



*Figure 2. `setuid()` parameters AMODE 31 and AMODE 64*

The other callable services can be divided into three groups:

- Callable services that have doubleword instead of fullword fields for parameter addresses in the 64-bit version**

Some callable services, such as `shmget()`, have doubleword instead of fullword fields for parameter addresses in the 64-bit version. As shown in [Figure 3 on page 10](#), the 64-bit version of the `shmget()` service has a change in the size of the second parameter. For the 31-bit version (BPX1MGT), the `Shared_Memory_Size` parameter is 4 bytes long. For the 64-bit version (BPX4MGT), the `Shared_Memory_Size` parameter is 8 bytes long, to accommodate a possible new size of 16 petabytes:

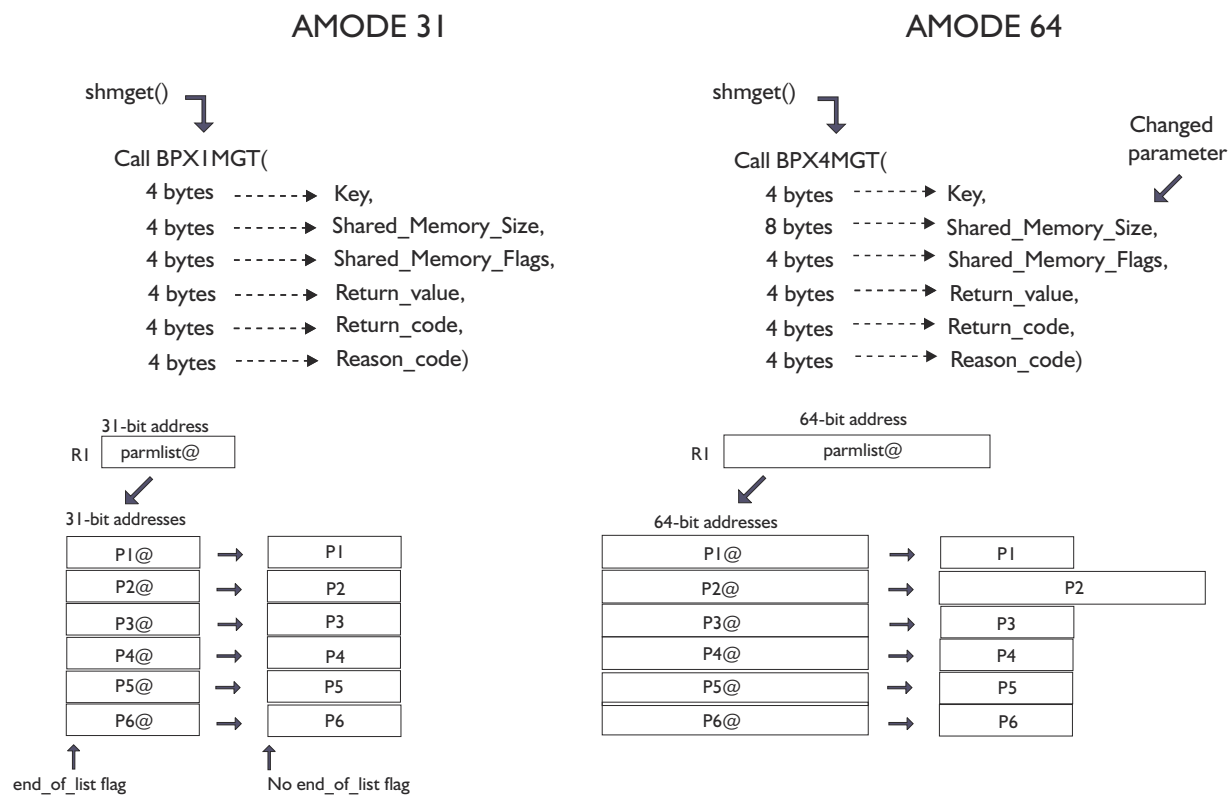


Figure 3. `shmget()` parameters AMODE 31 and AMODE 64

2. **Callable services that have an additional parameter in the 64-bit version, such as `loadhfs()`**

Other callable services, such as `loadhfs()`, have an additional parameter in the 64-bit version. Where addresses or lengths are passed back in the `Return_value` parameter, an 8-byte parameter is added to the 64-bit version of the service. (For compatability issues, the `Return_value`, `Return_code`, and `Reason_code` will always remain 4-byte fields.)

The 64-bit version of the `loadhfs()` service has an additional parameter. For `BPX1LOD`, the entry point address of the loaded executable is returned in the `Return_value` parameter. For `BPX4LOD`, the 64-bit entry point address is returned in the 8-byte `entry_point` parameter.



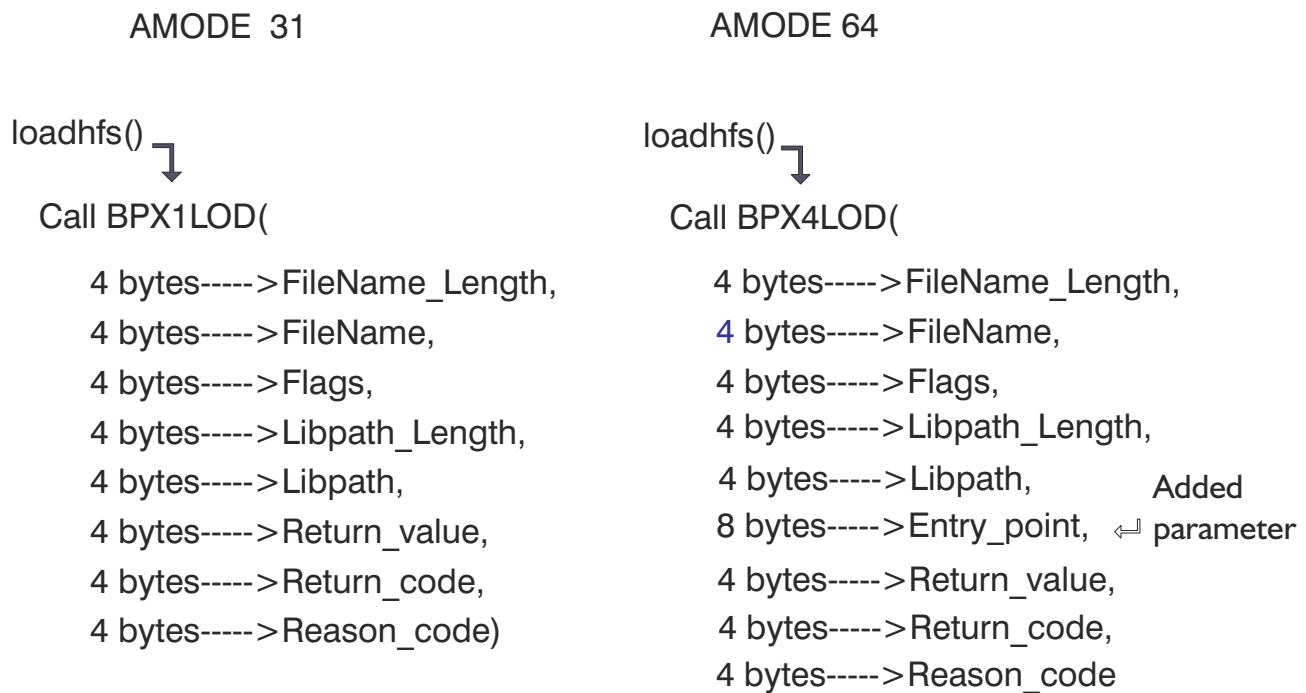


Figure 4. loadhfs() parameters AMODE 31 and AMODE 64

### 3. Callable services that use parameter structures with address fields that are at different offsets in the 64-bit version

A subset of callable services use parameter structures whose 64-bit address fields are at different offsets from their 31-bit counterparts; or whose address fields, because of their increased size, have caused other fields in the structure to be at different offsets for AMODE 64 callers. These are:

- BPX4AIO, with the AIOCB structure.
- BPX4RDV and BPX4WRV, with the iov structure.
- BPX4SMS and BPX4RMS, with the MSGH and iov structures.

For example, in the 64-bit version of the readv service, BPXB4RDV, the 64-bit address fields in the iov structure are at different offsets within the iov from their 31-bit counterparts:

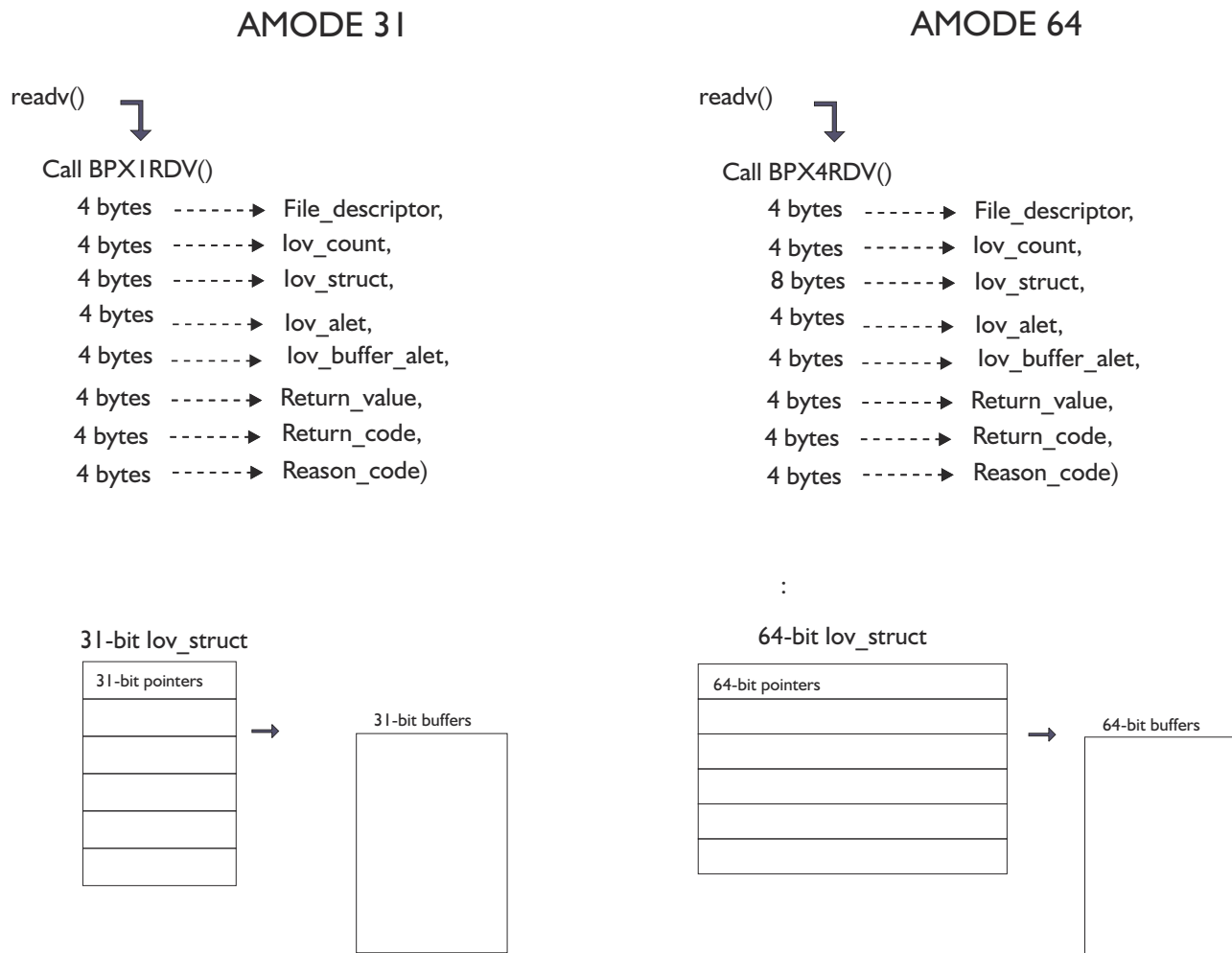


Figure 5. readv iov structure AMODE 31 and AMODE 64

The descriptions of the individual callable services contain detailed information about using the services in 64-bit AMODE.

## System control offsets

The offsets into the callable services table for the BPX4xxx calls are the same as the offsets for the BPX1xxx calls. The kernel reacts to the AMODE of the caller, which is saved in the linkage stack at the time of the PC to the kernel, and not to the stub. The kernel doesn't know whether a program is calling BPX1xxx or BPX4xxx. If you invoke a BPX4xxx stub in AMODE 31, the kernel will process the parameters of its BPX1xxx counterpart. If the parameters are different, an EFAULT exception will probably occur.

## Support for multiple AMODES in a single process

At the assembler level, the kernel supports multiple AMODES in a single process, and switching back and forth between AMODES on a single thread. Different threads in a single process can have different AMODES.

## Support for SRB callers

See Chapter 3, “Callable services for SRB mode routines,” on [page 1041](#) for the list of services that are supported for SRB mode callers running in AMODE 64.

## Chapter 2. Callable services descriptions

This topic describes each of the callable services. These services are arranged in alphabetic order. A sample invocation of each service is in [Appendix D, “Callable services examples—AMODE 31,”](#) on page 1219.

If you are unfamiliar with the conventions used to describe the system calls, refer to [Chapter 1, “Invocation details for callable services,”](#) on page 1.

### accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket

#### Function

The accept callable service allows a server to accept a connection request from a client. It extracts the first connection on the queue of pending connections, creates a new socket with the same properties as the specified socket, and allocates a new descriptor for that socket. If there are no connections pending, the service either blocks until a connection request is received, or fails with an EWOULDBLOCK, depending on whether the specified socket is marked as blocking or nonblocking.

#### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1ACP)</b>	31-bit task or SRB mode
<b>AMODE (BPX4ACP)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

#### Format

The syntax format is as follows:

```
CALL BPX1ACP,(Socket_descriptor,
               Sockaddr_length,
               Sockaddr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ACP with the same parameters.

#### Parameters

**Socket\_descriptor**  
Supplied parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword that contains the socket file descriptor for which the accept is to be done.

**Sockaddr\_length**  
Supplied and returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a field that contains the length of Sockaddr. On return, this field specifies the size required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information contained in Sockaddr is truncated to the length supplied on input. The field can be zero if no value is passed for Sockaddr. The size of the field should be less than 4096 bytes (4KB) in length.

**Sockaddr**  
Supplied and returned parameter

**Type:**  
Structure

**Length:**  
Length specified by Sockaddr\_length

The name of a field that contains the socket address of the connecting client. The format of Sockaddr is determined by the domain in which the client resides. This field can be null if the caller is not interested in the client address. For more information about the format of this structure, see [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#).

**Return\_value**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the accept service returns one of the following:

- A socket descriptor, if the request is successful.
- -1, if the request is not successful.

**Return\_code**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the accept service stores the return code. Return\_code is returned only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\) in z/OS UNIX System Services Messages and Codes](#). The following table lists the values that can be returned.

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, or JRFileNotOpen.

Return_code	Explanation
<b>ECONNABORTED</b>	Software-caused connection termination.
<b>EINTR</b>	A signal interrupted the accept service before any connections were available. The following reason code can accompany the return code: JRSignalReceived.
<b>EINVAL</b>	The socket is not accepting connections. A listen must be done prior to the accept. The following reason code can accompany the return code: JRListenNotDone.
<b>EIO</b>	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
<b>ENFILE</b>	Too many files are open in the system. The following reason code can accompany the return code: JRMaxSockets.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EOPNOTSUPP</b>	The referenced socket is not a type that supports the requested function.
<b>EWouldBlock</b>	The socket file descriptor is marked nonblocking, and no connections are present to be accepted.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the service stores the reason code. Reason\_code is returned only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The Socket\_descriptor that is passed refers to the socket that was created with the socket callable service, was bound to an address with the bind callable service, and that has issued a successful call to the listen callable service.  
  
Before calling the accept service, you can find out if the socket is pending a connection by doing a read select with the select callable service.
2. In order for Sockaddr to be returned for a UNIX domain socket, the client application doing the connect must bind a unique local name to the socket using the bind service before running the connect service.
3. See [Chapter 3, “Callable services for SRB mode routines,”](#) on [page 1041](#) for more information about programming considerations for SRB mode.

**Related services**

- [“asyncio \(BPX1AIO, BPX4AIO\) — Asynchronous I/O for sockets”](#) on [page 32](#)
- [“bind \(BPX1BND, BPX4BND\) — Bind a unique local name to a socket descriptor”](#) on [page 76](#)
- [“listen \(BPX1LSN, BPX4LSN\) — Prepare a server socket to queue incoming connection requests from clients”](#) on [page 391](#)

- [“select/selectex \(BPX1SEL, BPX4SEL\) — Select on file descriptors and message queues ” on page 706](#)
- [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets ” on page 865](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1ACP \(accept\) example” on page 1220](#).

# accept4 (BPX1AC4, BPX4AC4) — Accept a connection request from a client socket and set flags

---

## Function

The accept callable service allows a server to accept a connection request from a client. It extracts the first connection on the queue of pending connections, creates a new socket with the same properties as the specified socket, and allocates a new descriptor for that socket. If there are no pending connections, the service blocks until a connection request is received, or fails with an EWOULDBLOCK, depending on whether the specified socket is marked as blocking or nonblocking. With the accept4 callable service, you can also set the SOCK\_CLOEXEC and SOCK\_NONBLOCK flags.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PI2)</b>	31-bit
<b>AMODE (BPX4PI2)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1AC4( socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4AC4 with the same parameters.

## Parameters

### socket\_descriptor

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the socket descriptor for which the accept is to be done.

### Sockaddr\_length

Supplied and returned parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a field that contains the length of Sockaddr. On return, this field specifies the size that is required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information that is contained in Sockaddr is truncated to the length supplied on input. The field can be zero if no value is passed for Sockaddr. The size of the field should be less than 4096 bytes (4 KB) in length.

### Sockaddr

Supplied and returned parameter.

#### Type

Integer.

#### Length

Length specified by Sockaddr\_length.

The name of a field that contains the socket address of the connecting client. The format of Sockaddr is determined by the domain in which the client resides. This field can be null if the caller is not interested in the client address. For more information about the format of this structure, see [“BPXYSOCK — Map SOCK-ADDR structure and constants”](#).

## Flags

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that indicates the Initialization options.

Flag	Value	Description
<b>O_CLOEXEC</b>	'00001000'X	Sets the close-on-exec (FD_CLOEXEC) flag on the two new file descriptors.
<b>O_NONBLOCK</b>	'00000004'X	Set the O_NONBLOCK file status flag on the open file descriptions referred to by the new file descriptors. Using this flag saves extra calls to fcntl() to achieve the same result.

### Return\_value

Returned parameter.

#### Type

Integer

**Length**

Fullword

The name of a fullword in which the accept4 initialization service returns the accept file descriptor that was created, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the accept4 service stores the return code. The accept4 service returns the Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The accept4 service returns the same return codes as the accept syscall with an additional return code.

**Return\_code****Explanation****EINVAL**

The Options parameter does not specify a valid value.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the accept4 service stores the reason code. It returns the Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

For more information, see [“accept \(BPX1ACP, BPX4ACP\) — Accept a connection request from a client socket”](#) on page 13.

**Usage notes**

1. The Socket\_descriptor that is passed refers to the socket that was created with the socket callable service, was bound to an address with the bind callable service, and that has issued a successful call to the listen callable service.  
  
Before calling the accept service, you can find out whether the socket is pending a connection by doing a read select with the select callable service.
2. In order for Sockaddr to be returned for a UNIX domain socket, the client application handling the connect must bind a unique local name to the socket using the bind service before running the connect service.
3. For more information about programming considerations for SRB mode, see [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041

**Related services**

- [“asyncio \(BPX1AIO, BPX4AIO\) — Asynchronous I/O for sockets”](#) on page 32
- [“bind \(BPX1BND, BPX4BND\) — Bind a unique local name to a socket descriptor”](#) on page 76
- [“listen \(BPX1LSN, BPX4LSN\) — Prepare a server socket to queue incoming connection requests from clients”](#) on page 391
- [“select/selectex \(BPX1SEL, BPX4SEL\) — Select on file descriptors and message queues”](#) on page 706



- [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets ” on page 865](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1AC4 \(accept4\) example” on page 1220](#) and [“BPX4AC4 \(accept4\) example” on page 1314](#).

# accept\_and\_recv (BPX1ANR, BPX4ANR) — Accept a connection and receive the first block of data

---

## Function

The accept\_and\_recv callable service accepts the next connection on a socket and receives the first block of data. The new socket's descriptor, the peer's remote address, and the caller's local address are also returned. The service does not return until some data has arrived.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1ANR)</b>	31-bit
<b>AMODE (BPX4ANR)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax has the following format:

```
CALL BPX1ANR, (Socket_desc,
               Accepted_socket,
               Remote_addr_len,
               Remote_addr,
               Local_addr_len,
               Local_addr,
               Buffer_len,
               Buffer,
               Buffer_alet,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ANR with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## **Parameters**

### **Socket\_desc**

Supplied parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword that contains the socket descriptor for which the accept\_and\_recv() is to be done. This is the server's "listen socket."

### **Accepted\_socket**

Supplied and returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword that contains one of the following on input:

- -1, indicating that the system is to assign a new descriptor to the accepted connection. The new descriptor is returned in this parameter. Note that a valid accepted socket descriptor is returned for partial success cases.
- If supported by the system, the value of a reusable socket descriptor with which the accepted connection is to be associated. Socket descriptors are reused after they have been used on a send\_file that specified SF\_REUSE. Reusable socket descriptors are created initially through an accept or an accept\_and\_recv. (See [“send\\_file \(BPX1SF, BPX4SF\) — Send a file on a socket”](#) on page 730.)

The SF\_REUSE option is not supported for send\_file.

### **Remote\_addr\_len**

Supplied and returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword (doubleword) that contains the length of Remote\_addr. This field is updated with the length of the socket address that is returned in Remote\_addr. If you do not want the Remote\_addr, specify 0 for Remote\_addr\_len.

### **Remote\_addr**

Supplied and returned parameter

#### **Type:**

Structure

#### **Length:**

Remote\_addr\_len

The name of an area that contains the sockaddr structure that is returned for the client that is connecting.

### **Local\_addr\_len**

Supplied and returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword (doubleword) that contains the length of Local\_addr. This field is updated with the length of the socket address that is returned in Local\_addr. If you do not want the Local\_addr, specify 0 for Local\_addr\_len.

#### **Local\_addr**

Supplied and returned parameter

##### **Type:**

Structure

##### **Length:**

Local\_addr\_len

The name of an area that contains the sockaddr structure that is returned for the server's port on which the connection arrives.

#### **Buffer\_len**

Supplied parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword that contains the length of Buffer. If this value is zero, no receive is done, and the accept\_and\_recv request completes when a connection is available.

#### **Buffer**

Returned parameter

##### **Type:**

Area

##### **Length:**

Buffer\_len

The name of an area that contains the received data.

#### **Buffer\_alet**

Supplied parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a field that contains the alet of the buffer. For buffers in the caller's primary address space, this value should be 0.

#### **Return\_value**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the accept\_and\_recv service returns one of the following:

- The number of bytes (zero or greater) that are received into the buffer, if the request is successful. Zero bytes can occur if the client closed the socket without sending any data, if a value of zero was specified for Buffer\_len, or if no data was received within the active timeout interval. See [“Usage notes” on page 23](#) for more information.
- -1 with a Return\_code of EINTRNODATA, if the request was interrupted by a signal in the time between the arrival of the connection and the arrival of the first data. The connection is established, and Accepted\_socket returns the new socket descriptor.

- -1 with a Return\_Code of EWOULDBLOCK, if the request was interrupted because the SO\_RCVTIMEO value expired before data was received. The connection is established, and Accepted\_socket returns the new socket descriptor.
- -1 with any other Return\_Code, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the accept\_and\_recv service stores the return code. The Return\_code is returned only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The following table lists the values that can be returned in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBADF</b>	A file descriptor that was not valid was supplied. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	The connection was reset by a peer. The following reason code can accompany the return code: JRSockNotCon.
<b>ECONNABORTED</b>	The connection has been dropped.
<b>EFAULT</b>	An address that was passed cannot be accessed in the key of the caller.
<b>EINTR</b>	A signal interrupted the accept_and_recv service before a connection had arrived. The following reason code can accompany the return code: JRSignalReceived.
<b>EINTRNODATA</b>	A signal interrupted the accept_and_recv service after a connection had been established but before any data had arrived. This is a partial success, and the session has been established. A new socket descriptor is returned in Accepted_socket.
<b>EIO</b>	An I/O error occurred on one of the descriptors.
<b>EINVAL</b>	The socket is not accepting connections.
<b>EISCONN</b>	Accepted_socket is either bound or already connected.
<b>EMFILE</b>	OPEN_MAX descriptors are currently open in the calling process.
<b>ENOBUFS</b>	The service could not obtain a buffer. The following reason code can accompany the return code: JROutOfSocketCells.
<b>ENOMEM</b>	The service could not obtain memory to complete the operation.
<b>ENOREUSE</b>	Socket descriptor reuse is not supported.
<b>ENOSR</b>	Insufficient STREAMS resources were available for the operation to complete.
<b>ENOTSOCK</b>	Socket_desc does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EOPNOTSUPP</b>	The socket type of the specified socket does not accept connections; or O_NONBLOCK is set for this socket. Nonblocking mode is not supported for this function.

**Return\_code****Explanation****EWOULDBLOCK**

A new connection has been established, but the SO\_RCVTIMEO timeout value was reached before data was available. This is a partial success, and the session has been established. A new socket descriptor is returned in Accepted\_socket.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the accept\_and\_recv service stores the reason code. The Reason\_code is returned only if Return\_value is --1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Nonblocking mode is not supported for this function. If O\_NONBLOCK is set on the Socket\_desc parameter, the function fails with an EOPNOTSUPP error.
2. If accept (BPX1ACP, BPX4ACP) and accept\_and\_recv (BPX1ANR, BPX4ANR) calls are both used on the same socket, it cannot be predicted which calls will be satisfied and in which order. Note also that a mixture of accept and accept\_and\_recv is discouraged as it may result in a reduced performance benefit that is achieved using accept\_and\_recv exclusively.
3. SO\_SNDTIMEOUT and SO\_RCVTIMEOUT values are propagated from the server to the new, accepted connections.
4. If SO\_RCVTIMEOUT is specified on the server socket, the timeout for new connections is started when the connection is first established, rather than when the BPX1ANR service is issued. When the RCV\_TIMEOUT occurs the service completes with a Return\_code of -1 and Reason\_Code of EWOULDBLOCK. A new or reused socket descriptor is returned in Accepted\_Socket.
5. If SO\_RCVTIMEOUT is not specified on the server socket, an internal timer is started when the new connection is first established. If data from the client is not received within the internal timeout interval, the BPX1ANR completes successfully with a Return\_Value of zero. A new or reused socket descriptor is returned in Accepted\_Socket.
6. The accept\_and\_recv function is designed to work with the send\_file function to provide an efficient file transfer capability for a connection-oriented server with short connection times and high connection rates.
7. For more information about programming considerations for SRB mode, see [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041.

**Related services**

- [“accept \(BPX1ACP, BPX4ACP\) — Accept a connection request from a client socket”](#) on page 13
- [“recv \(BPX1RCV, BPX4RCV\) — Receive data on a socket and store it in a buffer”](#) on page 680

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1ANR \(accept\\_and\\_recv\) example”](#) on page 1222.

## access (BPX1ACC, BPX4ACC) – Determine if a file can be accessed

### Function

The access callable service determines whether the caller can access a file. You identify the file by its path name.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1ACC)</b>	31-bit
<b>AMODE (BPX4ACC)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax has the following format:

```
CALL BPX1ACC, (Pathname_length,
               Pathname,
               Access_mode,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ACC with the same parameters.

### Parameters

#### Pathname\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the pathname of the file.

#### Pathname

Supplied parameter

#### Type

Character string

#### Character set:

No restriction

#### Length

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file to be checked for accessibility. The length of this field is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory. The search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

### **Access\_mode**

Supplied parameter

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword field that indicates the access mode to be tested. This field is mapped by the BPXYACC macro. The values for this field are:

#### **ACC\_F\_OK**

Test for file existence. This is the default value.

#### **ACC\_R\_OK**

Test for permission to read.

#### **ACC\_W\_OK**

Test for permission to write.

#### **ACC\_X\_OK**

Test for permission to execute or search.

#### **ACCWAIT**

If an asynchronous mount is in progress, wait for it to complete.

#### **ACCDEVNO**

Return the devno of the file in Return\_value.

#### **ACCEFFID**

Use the effective ID rather than the real ID to check for permission.

### **Return\_value**

Returned parameter

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword in which the access service returns 0 if the request completes successfully (that is, the file exists or access is permitted), or -1 if the request is not successful, or the file cannot be accessed in the specified way.

### **Return\_code**

Returned parameter

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword in which the access service stores the return code. It returns Return\_code only if Return\_value is -1. For a list of return code values, see *Return codes (errno)* in *z/OS UNIX System Services Messages and Codes*. The Return\_code parameter returns one of the values listed in the following table.

Return_code	Explanation
<b>EACCES</b>	The calling process does not have appropriate permissions to access the file in the ways specified by the Access_Mode parameter, or does not have search permission for some component of the Pathname prefix.
<b>EINVAL</b>	The Access_Mode parameter is incorrect. The following reason code unique to the access service can accompany the return code: JRInvalidAMODE.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or some component of the pathname is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file named Pathname was found, or no Pathname was specified. The following reason code unique to the access service can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	A component of the Pathname prefix is not a directory.
<b>EROFS</b>	The Access_Mode parameter is testing for write access to a read-only file system. The following reason code unique to the access service can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword where the access service stores the reason code. The service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. Testing for file permission is based on the real user ID (UID) and real group ID (GID), unless the ACCEFFID bit has been set on. In that case, the effective ID is used for the test.
2. The caller can test for the existence of a file, or for access to the file, but not for both.
3. In testing for permission, the caller can test for any combination of read, write, and execute permission. If the caller is testing a combination of permissions, Return\_value indicates failure if any one of the accesses is not permitted.
4. If the caller has appropriate privileges (see “Authorization” on page 8), the access test is successful even if the permission bits are off, except when testing for execute permission. When the caller tests for execute permission, at least one of the execute permission bits must be on for the test to be successful.
5. If the Access\_mode parameter is zero, the service performs the existence test, ACC\_F\_OK.

**Related services**

- “chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory” on page 101
- “faccessat (BPX1FAA, BPX4FAA) — Checks user's permissions of a file relative to a directory file descriptor” on page 186
- “open (BPX1OPN, BPX4OPN) — Open a file” on page 511



- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1ACC \(access\) example” on page 1219](#).

# aio\_suspend (BPX1ASP, BPX4ASP) — Wait for an asynchronous I/O request

---

## Function

The aio\_suspend callable service suspends the calling thread until a specified asynchronous I/O event, specified timeout, or signal occurs.

## Requirements

Operation	Environment
<b>Authorization</b>	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1ASP)</b>	31-bit
<b>AMODE (BPX4ASP)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax has the following format:

```
CALL BPX1ASP, (Aiocb_Ptr_List,
               Aiocb_Ptr_Count,
               Seconds,
               Nanoseconds,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ASP with the same parameters.

## Parameters

### Aiocb\_Ptr\_List

Supplied parameter

### Type

Structure

**Length**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of an Aiocb. Each Aiocb represents a previously submitted asynchronous I/O operation that the thread is to wait on for completion. The number of Aiocb pointers in the list is represented by the Aiocb\_Ptr\_Count parameter.

**Aiocb\_Ptr\_Count**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the number of pointers in the Aiocb\_Ptr\_List. If you do not want to wait on any asynchronous I/O requests, define Aiocb\_Ptr\_Count as the name of a fullword that contains 0.

**Seconds**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains an unsigned integer that is the maximum number of seconds the calling program will wait for one of the specified asynchronous I/O events to occur.

**Note:**

1. Seconds can be any value greater than or equal to 0 and less than or equal to 4 294 967 295.
2. The Seconds and Nanoseconds values are combined to determine the timeout value. A combined value of zero indicates that the aio\_suspend service will not wait at all. A value of AIO#NO\_ASP\_TIMEOUT indicates that no timeout value is set. (See [“BPXYAIO — Map asyncio parameter list”](#) on page 1055.)

**Nanoseconds**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains an unsigned integer that is the number of nanoseconds to be added to the value specified by the Seconds parameter.

**Notes:**

1. Nanoseconds can be any value greater than or equal to 0 and less than or equal to 1 000 000 000.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the aio\_suspend service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the aio\_suspend service stores the return code. Return\_code is returned only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The aio\_suspend service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	One or more of the specified parameters are not valid. The following reason codes unique to the aio_suspend service can accompany the return code: JrNanoSecondsTooBig, JrMaxAiocbECB.
<b>EFAULT</b>	One of the specified parameters contains the address of a storage area that is not accessible to the caller. The following reason codes unique to the aio_suspend service can accompany the return code: JrOK, JrBadAioEcb.
<b>EINTR</b>	The service was interrupted by a signal. One or more of the specified asynchronous I/O requests might have completed.
<b>EAGAIN</b>	The service timed out before any of the specified asynchronous I/O requests had completed.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the aio\_suspend service stores the reason code. Reason\_code is returned only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The number of pointers to Aiocbs that use application-supplied ECB pointers for invocations of the aio\_suspend service is limited to 254 when the Seconds and Nanoseconds parameters are both set to zero, and to 253 if either is nonzero. See [“asyncio \(BPX1AIO, BPX4AIO\) — Asynchronous I/O for sockets”](#) on page 32 for information on how to supply user-defined ECBs in the Aiocb data area.
2. If the Aiocbs are specified without application-supplied ECB pointers, there is no limit on the number of Aiocb pointers.
3. The Aiocbs that are represented by the list of Aiocb pointers must reside in the same storage key as the caller of the aio\_suspend service. If the Aiocb Pointer List or any of the Aiocbs represented in the list are not accessible by the caller, an error of EFAULT may occur.
4. Aiocb pointers in the list with a value of zero are ignored.
5. A timeout value of zero (Seconds + Nanoseconds) means that the aio\_suspend service does not wait at all, but checks for any completed asynchronous I/O requests. If it finds none, it returns with an error of EAGAIN; otherwise, it returns with a Return\_value of 0.
6. A passed timeout value of AIO#NO\_ASP\_TIMEOUT (see [“BPXYAIO — Map asyncio parameter list”](#) on page 1055) means that no timeout value is set. The aio\_suspend service waits until an asynchronous I/O request completes or until a signal is received.

7. The AioCBS that are passed to the aio\_suspend service must not be freed or reused by other threads in the process while this service is still in progress. The service may use the AioCBS even after the asynchronous I/O completes. This restriction prevents multiple threads from doing aio\_suspend()s on the same AioCb at the same time. The results of modifying the AioCb during an aio\_suspend are unpredictable.
8. If the aio\_suspend service is being called in AMODE 64 (BPX4ASP), the AioCb\_Ptr\_List must contain 64-bit pointers only.

**Related services**

- [“asynCIO \(BPX1AIO, BPX4AIO\) — Asynchronous I/O for sockets” on page 32](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1ASP \(aio\\_suspend\) example” on page 1222](#).

## alarm (BPX1ALR, BPX4ALR) — Set an alarm

---

**Function**

The alarm call generates a SIGALRM signal after the number of seconds specified by the Seconds parameter have elapsed. The SIGALRM signal delivery is directed to the calling thread.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1ALR):</b>	31-bit
<b>AMODE (BPX4ALR):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

AMODE 64 callers use BPX4ALR with the same parameters.

**Format**

The syntax has the following format:

```
CALL BPX1ALR, (Seconds,
               Return_value)
```

## Parameters

### Seconds

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of an unsigned fullword that contains the minimum number of seconds that are to pass between receipt of this request and generation of the SIGALRM signal. If the value is zero, any outstanding alarm request is canceled; no new alarm call time is set. Processor scheduling delays can cause the delivery of the SIGALRM signal to occur after the desired time.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of an unsigned fullword return value field. If there is a previous alarm request with time remaining, the alarm service returns a nonzero value that is the number of seconds until the previous request would have generated a SIGALRM signal. The return value is rounded to the nearest second, except when the time remaining is less than a half second. When the remaining time is less than a half second and greater than zero, Return\_value is set to 1. If there is no previous alarm request with time remaining, Return value is set to zero.

## Usage notes

1. The alarm service is always successful, and no return value is reserved to indicate an error.
2. An abnormal end is generated when failures are encountered that prevent the alarm service from completing successfully.
3. Alarm requests are not stacked; only one SIGALRM generation is scheduled in this manner. If SIGALRM was not generated, the call reschedules the time that SIGALRM is generated.

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“sleep \(BPX1SLP, BPX4SLP\) — Suspend execution of a process for an interval of time ” on page 859](#)

## Characteristics and restrictions

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413.](#)

## Examples

See [“BPX1ALR \(alarm\) example” on page 1221.](#)

## MVS-related information

Both the alarm service (BPX1ALR or BPX4ALR) and the sleep service (BPX1SLP or BPX4SLP) use the MVS STIMER macro. It is possible that two STIMER SET requests can be set by the alarm service and

the sleep service. If the task invokes both STIMERM SET and the alarm service, the limit of concurrent STIMERM SET requests for a task can be exceeded, which results in an abnormal end.

## asyncio (BPX1AIO, BPX4AIO) – Asynchronous I/O for sockets

### Function

The asyncio callable service performs I/O operations against a socket asynchronously. It also provides synchronous operations for compatibility with the regular functions.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1AIO)</b>	31-bit
<b>AMODE (BPX4AIO)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax has the following format:

```
CALL BPX1AIO,(Aiocb_length,
               Aiocb,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4AIO with the same parameters. All addresses in the Aiocb structure are doublewords.

### Parameters

#### Aiocb\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Aiocb control block that is being passed in the next parameter. To determine the value of Aiocb\_length, use the BPXYAIO macro (see [“BPXYAIO — Map asyncio parameter list”](#) on page 1055).

#### Aiocb

Supplied parameter and returned parameter.

#### Type

Structure

### Length

Specified by the Aiocb\_length parameter.

The name of an Aiocb structure to be used to control this I/O operation. For information about setting the Aiocb fields, see [“Usage notes for Aiocb control blocks”](#) on page 35.

The BPXYAIO macro (see [“BPXYAIO — Map asyncio parameter list”](#) on page 1055) maps the Aiocb.

### Return\_value

Returned parameter.

### Type

Integer

### Length

Fullword

The name of a fullword in which the asyncio service returns the results of the request:

- 0 indicates that an asynchronous request was successfully scheduled.  
When the I/O completes, the return value, return code, and reason code of the requested function are returned in the Aiocb, and the application is notified. For more information about asynchronous input/output, see [“Asynchronous input/output”](#) on page 35.
- -1 indicates that the system could not schedule the request, or the request itself failed immediately, for reasons such as parameter errors. Refer to Return\_code and Reason\_code for more details. There is no I/O completion notification.  
When the I/O function itself is rejected immediately, the return code and reason code are specific to that function. They are documented with the description of the regular version of the function.
- +1 indicates that the operation successfully completed synchronously. One of the following actions occurred:
  - AioOk2CompImd is specified, and the operation can be completed immediately.
  - AioSync is specified.
  - The function is Aio#Cancel, and AioCancelNoWait is not specified.

The system returns the return value, return code, and reason code of the requested function in the Aiocb. There is no I/O completion notification.

**Note:** These values are returned upon successful completion only. Immediate failures are always reported with a Return\_value of -1.

### Return\_code

Returned parameter.

### Type

Integer

### Length

Fullword

The name of a fullword in which the asyncio service stores the return code. The return code is returned only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The asyncio service returns one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	One of the following occurred: <ul style="list-style-type: none"> <li>The maximum number of queued signals was exceeded for this process (JrMaxQueuedSigs). This limit is specified with the MAXQUEUEDSIGS parameter of the BPXPRMxx parmlib member.</li> <li>The maximum number of outstanding asynchronous requests that are permitted for this process was exceeded (JrMaxAsyncIO). The Async I/O maximum is twice the sum of MAXQUEUEDSIGS and the process's file limit. The file limit is taken from RLIMIT_NOFILE or the BPXPRMxx MAXFILEPROC parameter.</li> </ul>
<b>EALREADY</b>	The AioCb was canceled.
<b>EBADF</b>	The AioFd field does not contain a valid descriptor or the descriptor of a socket. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EFAULT</b>	A supplied data area cannot be referenced.
<b>EINVAL</b>	A parameter is not valid. For example, AioBuffSize is negative, or AioCmd or AioNotifyType are unsupported values. The following reason codes can accompany the return code: JrAsyncBadAioCbLen, JrAsyncBadOffset, JrAsyncBadNotifyType, JrAsyncBadMsgHdrLen, JrAsyncBadSockAddr, JrAsyncBadCmd.
<b>EIO</b>	There was a network or transport failure.
<b>EMVSNINITIAL</b>	Support for unauthorized user exits failed to initialize.
<b>ENOSYS</b>	The socket transport or physical file system does not support asynchronous I/O. Possible value: JrAsyncOpNotSupp.
<b>EOPNOTSUP</b>	If the return code is JrMsgFlagInvalidFlag, the TCPIP stack does not support the specified AioPosixFlags value. If the return code is JrAsyncAnr, a previous accept() operation was processed on this server socket, and because of this the use of asynchronous accept_and_receive is not supported.
<b>EPERM</b>	The caller is not authorized. Consult Reason_code to determine the exact reason that the error occurred. If the return code is JrAsyncAuthErr, one of the following flags was set by an unauthorized caller: AioCallB4, AioUseUserKey, or AioCommBuff.

Refer also to the regular versions of the various functions for errors that might be detected before the system schedules the request, or if the request was processed asynchronously (AioSync was specified).

#### Reason\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the asyncio service stores the reason code. Reason\_code is returned only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

The following reason codes might be reported: JrAsyncAuthErr, JrAsyncBadAioCbLen, JrAsyncBadCmd, JrAsyncBadMsgHdrLen, JrAsyncBadNotifyType, JrAsyncBadOffset, JrAsyncBadSigNo, JrAsyncBadSockAddr, JrAsyncExitModeTCB, JrAsyncOpNotSupp,



JrAsyncSigKey0Err, JrReadUserStorageFailed, JrWriteUserStorageFailed, JrSyscallAbend, JrMsgInvalidFlag, JrAsyncANR.

## Asynchronous input/output

The asyncio service provides the capability to asynchronously perform those functions that are potentially blocking. These include the accept, connect, and receive and send types of functions.

The general flow of an asynchronous request is as follows:

1. All the parameters that are normally used on the regular version of the function are specified through the AioCb structure. Parameters necessary to control the features of an asynchronous request are also specified here.
2. After some preliminary checking, the system schedules the request and returns control to the caller. The AioRc field is set to EINPROGRESS. The application is free to continue with other work until it is notified that the I/O has completed. For more information about Notification, see [“Usage notes for AioCb control blocks”](#) on page 35.

I/O completion usually occurs under the following conditions:

- For reads, data is available or arrives from the network.
- For writes, system buffers are available to hold the caller's data. This is the point at which the caller's buffers can be reused or freed. It does not imply anything about the progress of the actual data transmission.
- For accept, a connection request is available or arrives.
- For connect, this depends on the socket type and specific transport. It is usually the point at which you can start sending and receiving on the socket. This does not necessarily mean that the server has accepted this connection.
- For accept\_and\_recv, connection request and the initial data from the client are available.

The AioCb and any areas pointed to from the AioCb, such as a receive buffer, must remain valid until the I/O has completed.

3. When it can complete the I/O, the system schedules an SRB to the caller's address space to perform the following sequence:
  - a. Optionally, call the exit program for preprocessing. See AioCallB4 in this topic for more information.
  - b. If the operation was successful up to this point, transfer the I/O data from or to the caller's buffers.
  - c. Update the AioRv, AioRc, and AioRsn fields of the AioCb with the status of the operation.
  - d. Perform the I/O completion notification as specified by AioNotifyType and other fields in the AioCb. Only one type of notification is issued. If no notification is requested, the application can check the AioRc field periodically until it changes from EINPROGRESS.

The I/O completion notification is sent for these situations:

- Send a signal.
- Send a message.
- Call the exit program, passing the AioCb.
- Post the ECB.

The asyncio service supports AF\_INET and AF\_INET6 sockets; it cannot be used with AF\_UNIX sockets.

## Usage notes for AioCb control blocks

The values set into this control block control the asyncio operation. The BPXYAIO macro maps the AioCb. The caller is responsible for setting the following fields.

Field	Description
-------	-------------

## **Function-specific fields**

### **AioCmd**

Specifies the function to be performed:

- Aio#Accept for accept (BPX1ACP, BPX4ACP)
- Aio#Anr for accept\_and\_recv (BPX1ANR, BPX4ANR)
- Aio#Connect for connect (BPX1CON, BPX4CON)
- Aio#Read for read (BPX1RED, BPX4RED)
- Aio#Write for write (BPX1WRT, BPX4WRT)
- Aio#ReadV for readv (BPX1RDV, BPX4RDV)
- Aio#WriteV for writev (BPX1WRV, BPX4WRV)
- Aio#Recv for recv (BPX1RCV, BPX4RCV)
- Aio#Send for send (BPX1SND, BPX4SND)
- Aio#RecvFrom for recvfrom (BPX1RFM, BPX4RFM)
- Aio#SendTo for sendto (BPX1STO, BPX4STO)
- Aio#RecvMsg for recvmsg (BPX1RMS, BPX4RMS)
- Aio#SendMsg for sendmsg (BPX1SMS, BPX4SMS)
- Aio#SelPoll for select (BPX1SEL, BPX4SEL) or poll (BPX1POL, BPX4POL).
- Aio#Cancel to cancel a prior asyncio request. See the usage notes for more information about canceling operations.

For more information about their semantics and returned information, refer to the descriptions of the regular versions of these functions.

### **AioFd**

The socket descriptor.

### **AioBuffPtr**

The address of the buffer for the particular operation.

- For read/write, recv/send, recvfrom/sendto, and accept\_and\_recv, the address of the data buffer.
- For readv/writev, the address of the iov, BPXYIOV (see [“BPXYIOV — Map the I/O vector structure”](#) on page 1098).
- For recvmsg/sendmsg, the address of the msghdr, BPXYMSGH (see [“BPXYMSGH — Map the message header”](#) on page 1109).
- For selpoll, the address of a PollFD array, BPXYPOLL (see [“BPXYPOLL — Map poll syscall parameters”](#) on page 1124).
- For cancel, the address of the Aiocb to be canceled, or 0 to cancel all outstanding asyncio requests on the descriptor.

In 64-bit mode, AioBuffPtr is a doubleword pointer field, and is at a different offset within the Aiocb.

### **AioBuffSize**

Specifies the size of whatever AioBuffPtr points to.

- For read/write, recv/send, recvfrom/sendto, and accept\_and\_recv, the length of the data buffer.
- For readv/writev, the number of elements in the iov array.
- For recvmsg/sendmsg, the length of the msghdr.
- For selpoll, the number of elements in the PollFD array.
- For cancel, this field is ignored.

### **AioBuffAlet**

For read/write, recv/send, recvfrom/sendto, readv/writev, and accept\_and\_recv operations, this field contains the ALET of whatever is pointed to by AioBuffPtr. For all other operations, this field is ignored. See usage note [“18”](#) on page 47 for more information about using ALETs.

### **AioSockAddrPtr**

Contains the address of a sockaddr structure area in the caller's primary address space. The sockaddr contains the address of the remote partner.

The sockaddr structure itself is supplied to the sendto and connect functions and returned by the recvfrom, accept\_and\_recv, and accept functions.

In 64-bit mode, AioSockAddrPtr is a doubleword pointer field, and is at a different offset within the Aiocb.

### **AioSockAddrLen**

Contains the length of the sockaddr structure pointed to by AioSockAddrPtr.

This field is supplied to all functions that use AioSockAddrPtr. It is updated with the returned sockaddr length by the recvfrom, accept, and accept\_and\_recv functions. The following functions allow a value of 0 to be specified, indicating that no sockaddr structure is to be returned: Aio#Accept, Aio#ANR and Aio#Connect.

### **AioLocSockAddrPtr**

Contains the address of a sockaddr structure area in the caller's primary address space. In 64-bit mode, AioLocSockAddrPtr is a doubleword pointer field and is at a different offset within the Aiocb. The accept\_and\_recv function updates this field with the local sockaddr structure.

### **AioLocSockAddrLen**

Contains the length of the sockaddr structure asyncio pointed to by AioLocSockAddrPtr. If you do not want the local sockaddr structure, specify 0 for AioLocSockAddrLen.

### **AioAnrSocket**

Used by the accept\_and\_recv function. On input contains one of the following:

- -1, indicating that the system is to assign a new descriptor to the accepted connection. The new descriptor is returned in this parameter. A valid accepted socket descriptor is returned for partial success cases as defined by the accept\_and\_recv service. (See [“accept\\_and\\_recv \(BPX1ANR, BPX4ANR\) — Accept a connection and receive the first block of data” on page 19.](#))
- If supported by the system, the value of a reusable socket descriptor with which the accepted connection is to be associated. Socket descriptors are reused after they have been used on a send\_file that specified SF\_REUSE. Reusable socket descriptors are created initially through an accept, an accept\_and\_recv, or through the asyncio commands AIO#ACCEPT and AIO#ANR. (See [“send\\_file \(BPX1SF, BPX4SF\) — Send a file on a socket” on page 730.](#))

### **AioMsgIovAlet**

Specifies the ALET of the recvmsg/sendmsg msghdr's iov. See the usage notes for more information about using ALETs.

### **AioIovBufAlet**

Specifies the ALET of all buffers pointed to from the iov that is used with the readv/writv and recvmsg/sendmsg functions. See the usage notes for more information about using ALETs.

### **AioPosixFlags**

On input, contains the MSG\_FLAGS value for the recv/send, recvfrom/sendto, recvmsg/sendmsg, and accept\_and\_recv functions (such as MSG\_OOB and MSG\_PEEK). For more information about the format of this field, see [“BPXYMSGF — Map the message flags” on page 1107](#)

Several of the MSG\_FLAGS are particularly useful with asyncio:

#### **MSG\_CONNTERM**

The asyncio will not complete until the socket session ends. AioBuffSize must be set to 0 and the other MSG\_FLAGS cannot be used.

- This flag provides a way to be notified at the time that a TCP socket session ends, independent of any actual receive-type operations outstanding on the socket.
- Any other outstanding requests can also be completed at connection termination.

#### **MSG\_WAITALL**

The asyncio will not complete normally until the requested amount of data has arrived.

## MSG\_PEEK

Just the front of the data will be received, without being removed from the socket. For example, if your messages contain a fixed header that contains the length of the full message, just the header can be received asynchronously while the whole message is later received synchronously by the worker thread that will process the message. This also allows the asyncio to be done with a relatively small buffer and the worker thread will know how large a buffer needs to be allocated to hold the whole message.

## Asynchronous feature fields

### AioNotifyType

Specifies the type of asynchronous notification:

#### Aio#Posix

Sets the return value, code, and reason fields of the AioCb. Optionally, sends the signal specified in the AioSigEvent structure.

#### Aio#MVS

Sets the return value, code, and reason fields of the AioCb. Optionally, calls the exit program or posts the ECB.

#### Aio#MsgQ (AIO\_MSGQ)

Sets the return value, code, and reason fields of the AioCb, and sends the message specified in the AioMsgEvent structure.

Default: Aio#Posix.

### AioExitPtr

Specifies the address of a program that the system is to call when the I/O completes. The AioCb from the original request is passed to the exit. See [“4” on page 42](#) for details. In 64-bit mode, AioExitPtr is a doubleword pointer field, and is at a different offset within the AioCb.

### AioExitData

An eight-byte area that is reserved for use by the application and the exit program. The system does not inspect or change this area.

### AioExitModeTcb

For authorized TCB callers only, this specifies the mode (SRB or TCB) in which the exit program is to be called:

- 0 on an SRB in the caller's address space. This is the default.
- 1 on the caller's TCB.

For non-authorized callers, the exit is always run on the caller's TCB. If the caller's TCB ends before the I/O completes, the system does not run the exit.

### AioECBPtr

Specifies the address of an ECB in the caller's home address space that the system is to post when the I/O completes.

### AioSigEvent

For Aio#Posix, this is a SigEvent structure that controls the signal generation. It contains the following fields:

#### Sigev\_Notify

Set to Sigev\_Signal (0) to send the signal, or Sigev\_none (1) to not send any signal. Default: Sigev\_Signal.

#### Sigev\_Signo

Set to the signal number to be sent.

In 64-bit mode, SigEvent is a larger structure with several doubleword pointer fields; consequently, AioSigEvent is larger and at a different offset within the AioCb.

### AioMsgEvent

For Aio#MsgQ, this is a structure that specifies the I/O completion message. See [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue” on page 465](#) for more information about messages and

message queues. This structure overlays the AioSigEvent area and contains fields which correspond directly to the parameters in the msgsnd function:

- For AioMsgev\_QID, specifies the message queue identifier that will receive the message.
- For AioMsgev\_Addr/AioMsgev\_Addr64, specifies the address of the message buffer. This buffer is defined by the Msgbuf/Msgbuf64 structure in BPXYMSG and contains the message type and message text. For C programs the name of this address field is the same for both 31-bit and 64-bit amodes, and the msgbuf structure is user-defined as described in [msgsnd\(\) - Message send operations in z/OS XL C®/C++ Runtime Library Reference](#).
- For AioMsgev\_Size, specifies the length of the message text. This value does not include the length of the message type field that precedes the text in the message buffer. The message text length is limited to 240 bytes.
- For AioMsgev\_Flag, specifies the action to take if the message queue fills up. Specify 0 to wait, or IPC\_NOWAIT, defined in BPXYIPCP, to not wait.

### AioSiCode

A halfword signal code value that is to be associated with this I/O. This is only meaningful for I/O completion notification via signals. When the I/O completes and the completion signal is delivered through sigwaitinfo() or sigtimedwait(), the Si\_code field of the resulting siginfo structure contains the value specified here. The halfword specified here occupies the lower half of the fullword Si\_code. Normally the Si\_code is set to SI\_ASYNCIO# for asynchronous I/O completions, but if AioSiCode is not zero, that value will be used when the signal is sent. The meaning of this value is up to the application, and is not interpreted by the system.

### AioTimeout

A word that contains the timeout value for SelPoll.

- A value of Aio#Forever (0) means to wait forever. This is the default.
- A value of Aio#NoWaiting (-1) means to not wait at all

AioTimeout can also be used to set a time limit for other synchronous operations (AioSync), such as Aio#Recv, Aio#Accept, and Aio#ANR. The corresponding socket option, SO\_RCVTIMEO or SO\_SNDTIMEO, also needs to be set to a nonzero value using the setsockopt() callable service. The timeout value used is the lesser of the AioTimeout or the SO\_RCVTIMEO/SOCK#SO\_SNDTIMEO value specified.

The SO\_RCVTIMEO socket option can be used to achieve a timeout function for asynchronous receive type operations. For more information, see in [getsockopt\(\) - Get the options associated with a socket in z/OS XL C/C++ Runtime Library Reference](#).

### AioOk2CompImd

Specifies that the system may complete an asynchronous request immediately if it can do so without waiting, and without making any task switches. Otherwise, the system schedules the request for normal asynchronous processing.

If the request completes successfully and immediately, the return value from asyncio is 1. The system returns the results of the function itself in the Aiocb. In this case there is no I/O completion notification.

On an asynchronous read (or on accept\_and\_recv), if data has already arrived, this option avoids the extra overhead of scheduling the SRB and performing the notification. You must code the program to handle the received data in two places: after the call to asyncio and after the notification.

For the best performance, you should always set AioOk2CompImd and be able to handle I/O completion at the point of the call.

Default: Off, or it is not all right to complete immediately. The system issues the I/O completion notification.

### AioSync

Specifies that the system is to run the request synchronously. The caller will wait or block, as necessary, subject to the current value of the nonblocking state of the socket.

If the request is successful, the `Return_value` from `asyncio` is 1. The system returns the results of the function itself in the `Aiocb`. There is no I/O completion notification.

This option provides equivalence with the regular versions of the functions. It is useful for synchronous operations that must be cancelable, for operations whose waits should be limited by `AioTimeout`, and for calling the `select()` function with the much more efficient `poll()` interface.

Default: Off, or asynchronous.

**AioTcbAffinity**

Specifies that the I/O request should be canceled if the caller's TCB terminates. This field should be set if the `Aiocb` or buffer areas are in task-related storage and therefore will be freed when the task terminates.

Default: Off, or do not cancel the I/O for any type of I/O complete notification other than TCB Exit. For TCB Exits, the exit cannot be run after the TCB terminates, so the I/O will be canceled.

**AioCancelNoWait**

Specifies that a cancel operation is not to wait for all I/O completion notifications to finish before it returns to the caller. See usage note [“11” on page 45](#) for more information about canceling operations.

Default: Off, or wait.

**AioCancelNoNotify**

Specifies that a cancel operation is to skip the I/O completion notifications that have not already been issued. See usage note [“11” on page 45](#) for more information about canceling operations.

Default: Off, or issue the notifications.

**AioCallB4**

For authorized callers only. When on, this specifies that the exit program is to be called on the SRB for preprocessing before arrived data is transferred to the user's buffer. This provides a way to defer read buffer allocation until after the data has arrived. This call is in addition to the call that is made after the I/O has completed. See usage note [“4” on page 42](#) for more information about I/O completion exits. Usage note [“7” on page 43](#) has information about preprocessing.

Default: Off, or do not call the exit for preprocessing.

**AioUseUserKey**

For authorized callers only. When on, this specifies that the storage key in `AioUserKey` is to be used for all references to the functional parameters and data buffers. Only the `Aiocb` will be referred to with the caller's key.

Default: Off, or use the caller's key for all storage references.

**AioUserKey**

The key to be used for all references to the functional parameters and data buffers. This is only used when `AioUseUserKey` is on.

**AioCommBuff**

For authorized callers only. Specifies that the I/O buffers for this request reside in common storage, and may be addressed from any address space. Examples of common storage are ECSA and CADs data spaces. Having I/O buffers in common storage allows the system to copy data to them without having to have the caller's address space present, which improves the overall performance of asynchronous I/O. This flag is processed only for stream sockets; only for the operations of `Aio#Read`, `Aio#ReadV`, or `Aio#Recv`; and only with I/O complete notification via an SRB exit or an ECB. The `Aiocb` and the `iov` for `Aio#ReadV` do not have to be in common. It is critical that any outstanding I/O that has specified `AioCommBuff` be canceled explicitly before the descriptor is closed.

Default: Off, or the buffers reside in the caller's address space.

**AioACEE**

For SRB-mode callers only. Specifies the address of a security environment (ACEE) in the caller's home address space that is to be used for any multilevel security checks that may be done by TCP/IP during this I/O. This provides SRBs with a capability similar to the task-level security that is available

with the TCBSENV field of a task's TCB. This field is ignored for TCB-mode callers, for whom the TCBSENV field will be used if it is nonzero, and if multilevel security checks are necessary. Before the specified ACEE can be freed or invalidated, you must ensure that this I/O has completed or been canceled; or that the socket is closed.

Default: Off, or use task-level security.

The following fields, which are set by the system, pass back the results of the requested function, as defined for the regular version of that function:

- **AioRv**: Return\_Value
- **AioRc**: Return\_Code
- **AioRsn**: Reason\_Code

These fields are meaningful only after a successfully scheduled asynchronous request has completed, or when the asyncio service has a Return\_value of 1.

The AioRc field is set to EINPROGRESS when a request is successfully scheduled. This value is changed to reflect the final results of the operation when the I/O completes.

The AioRc field is set to ETIMEDOUT for any function that times out because the AioTimeOut field is used.

The AioRc field is set to ECANCELLED for a request that is subsequently canceled. See usage note [“11” on page 45](#) for more information about canceling operations.

**Note:** There are two ways to request that there be no notification: Assuming that the AioCb has been initialized to zeros, you can set Sigev\_Notify to Sigev\_none, or you can set AioNotifyType to Aio#MVS. If no notification is used, the program can occasionally check the AioRc field until it is no longer equal to EINPROGRESS.

Callers of BPX1AIO (BPX4AIO) are considered authorized if the program is running in supervisor state or a system key, or if the program is APF-authorized.

## Usage notes

1. See Chapter 3, [“Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.
2. Unauthorized callers are restricted in the following ways:
  - Exits are run on the caller's TCB. If that TCB ends before the I/O has completed, the exit is not run. There are restrictions on the use of exits.
  - Authorized TSO commands are not permitted while any asynchronous I/O is outstanding in a TSO address space.
  - The AioCallB4 and AioUseUserKey options are not available.
  - The AioCancelNoWait option is not available if exits are pending for the TCB from which the cancel call is made.
3. Using message queues for I/O completion notifications. These messages can be received with the BPX1QRC (msgrecv) function. If the message buffer address is 0, a default message is sent. The default message type will be SIGIO#, which is the number 23 and is defined in BPXYSIGH. The default message text will be eight bytes long and contain the AioCb address. For 31-bit callers the high word of the eight bytes is 0 and the aioCb address is in the lower word.

The program should be designed to respond to the message queue in a timely manner so that waiting is avoided. These messages are sent from a system SRB, and waiting will tie up this critical resource and impact overall performance. Using small messages and a large queue can also help avoid waiting. If IPC\_NOWAIT is specified and the queue fills up, or there is any other error in trying to send the message, the message will be lost. The system SRB will issue an EC6 abend with reason code FsAioMsgQError. The dump or logrec record that is produced will contain the return and reason codes from the internal call to BPX1QSN msgsnd() and potentially other diagnostic information about the problem.



The length of the message text is limited to 240 bytes.

4. You specify an exit program by setting AioNotifyType to Aio#MVS and putting the exit's address into the AioExitPtr field.

The exit is called in the AMODE of the caller, AMODE 31 for BPX1AIO and AMODE 64 for BPX4AIO.

The exit is passed the original Aiocb to correlate this completion with the original request. The Aiocb contains an application area (AioExitData) that can be used to communicate with the exit. Since the application allocates the Aiocb in the first place, the areas before and after the control block are also available for related use. The Aiocb can be embedded in a larger application control block that can easily be reached from the Aiocb.

The exit is usually called to process received data, or free the storage that has been tied up with this request. At this point, the Aiocb contains the final return value, return code, and reason code of the function.

When a request has been canceled, the AioRv field is -1, and the AioRc field is set to ECANCELED. See usage note “11” on page 45 for more information about canceling operations.

5. Authorized exits. When the calling program is authorized, the exit is run on a system SRB; it is authorized and in key 0.

Because the exit program is running on an SRB, the rules for SRB-mode callers must be followed if the exit makes any calls to z/OS UNIX functions. In particular, register 2 must be set before a call, to identify the process to which the exit belongs. See Chapter 3, “Callable services for SRB mode routines,” on page 1041 for details. The discussion on recovery for user SRBs is not relevant, because the exit is running on a z/OS UNIX SRB.

**Guideline:** There is an upper limit to the number of SRBs that are allowed to run at the same time. If this limit is reached, other I/O completions remain queued until an SRB becomes available.

The AioExitModeTCB flag can be used to run the exit on the original TCB rather than on the system SRB. In this case the restrictions that are listed for Unauthorized Exits apply. The exit is entered in the key of the first caller of BPX1AIO (BPX4AIO) in this process. This key cannot be changed.

6. Unauthorized exits When the calling program is not authorized, the exit is run on the caller's TCB in the caller's state and key. There are some restrictions:

- A C program calling BPX1AIO (BPX4AIO) with an exit specified must be POSIX(OFF). POSIX signal handling and POSIX threading, as provided by Language Environment®, are not supported for any task in the program's process.
- A program may not have invoked the BPX1MSS (BPX4MSS) service to register a signal interrupt routine.
- The exit program is not in any way an extension of the main program. A C exit must establish its own C environment on entry. (This is significantly different from C signal handling.) In order for the I/O interrupt to be delivered, the thread that calls BPX1AIO (BPX4AIO) must remain dubbed.
- All callers on all threads of a given process that are doing BPX1AIO (BPX4AIO) calls must be running with the same storage key.
- The I/O interrupt targets the RB that made the original call to BPX1AIO (BPX4AIO) for a given thread. If the target RB is not the top RB, the interrupt is deferred until the target RB becomes the top RB.
- A program must not have blocked the SIGIO signal, because the system uses this signal to schedule the exits.
- If the exit program ends abnormally, the system cleans up that request and continues with other exits that are waiting to be run on that TCB. There is no dump, and the originator of that request is not notified of the problem. To have these abends percolated to the TCB, so that the TCB's mainline recovery will be run or the TCB will be terminated, set the ThliTcbExitPerc bit on before you issue the call to BPX1AIO (BPX4AIO).

The exit is free to do whatever is supported within the environment from which it is called. It may issue another call to asyncio.



**Guideline:** The exit should not issue any blocking calls, and should not enter into long delays. This ties up the system SRB on which the exit is running. Unauthorized exits are blocking the TCB that made the original request.

7. Authorized callers can call the exit for preprocessing before the data is transferred, by using the AioCallB4 flag. This call is on the system SRB, and at a point before arrived data is moved into the application's receive buffer. It provides a way for the application to defer committing the necessary storage until just before it is actually needed.

This deferred allocation applies to the receive buffers only. All other structures that are related to the call must be present, and the total requested data length must be specified correctly. For read, recv, recvfrom, and accept\_and\_recv, AioBuffPtr may be 0; but AioBuffSize must be set to the amount that is being requested. For readv, the following conditions must be met:

- AioBuffPtr must point to an iov.
- AioBuffSize must contain a nonzero number of iov entries.
- The sum of the length fields in those entries must equal the amount being requested.
- The iov buffer pointers may be zero.

For recvmmsg, there must be a valid msghdr structure with its associated sockaddr area, and an iov structure as described for readv. You can specify a simple one-element iov on the initial call to carry the length information, and replace this with another iov to be used for the data transfer.

You cannot use deferred allocation with AioOk2CompImd.

When the preprocessing exit is called, the AioRv value usually contains the amount of data that is available, up to the requested amount. You can allocate smaller buffers when they will be sufficient. You should be prepared for cases in which AioRv is zero, when you should allocate buffers for the original requested amount. The actual amount of data that is received is returned in AioRv on the I/O completion call to the exit.

The preprocessing call is only made when the operation has, up to this point, been successful. The preprocessing call is in addition to, not a replacement of, the call that is made after the I/O completes. The exit can use the AioExitData area to record its entry, and thus distinguish between the first and second calls during a successful operation. If AioRv is -1, this is the only call that is made to the exit.

The preprocessing exit can change the following Aiocb fields to affect subsequent processing: AioBuffPtr, AioBuffAlet, AioExitPtr, and AioECBPTr. You may not change the function.

8. When the caller's process terminates:
  - Exits for requests that have not yet completed are not called.
  - SRB exits that are about to be called or are already running may continue to completion, with some exceptions. If the process' address space abnormally ends, the exit is not able to finish. If the exit suspends, or calls a system service that suspends, it can be abnormally ended with a 47B abend code. If you have recovery, you should not take a dump or write an error record for 47B abends, but retry and return.

SRB-mode routines should not call asyncio after it has entered process termination. See [Chapter 3, "Callable services for SRB mode routines,"](#) on page 1041.

9. The exit program receives control in the asyncio caller's address space and in the following environment:

Operation	Environment
<b>Authorization:</b>	Same as the caller of asyncio
<b>Dispatchable unit mode:</b>	SRB or TCB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE:</b>	Same as the caller of asyncio

## Operation

**ASC mode:**

**Interrupt status:**

**Locks:**

**Control parameters:**

## Environment

Primary mode

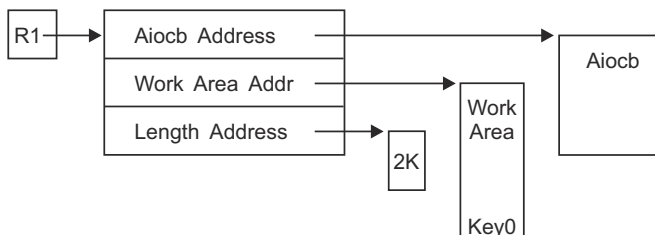
Enabled for interrupts

Unlocked

All parameters are addressable in the primary address space in key 0 storage.

On entry, register 1 points to a parameter list that contains:

- The address of the Aiocb that was specified on the asyncio call now completing
- The address of a 2K work area for the exit's use
- The address of the length of the work area



For authorized exits on the SRB, the work area is in key 0. For unauthorized exits, or authorized exits running on the TCB, it is in the key of the caller of asyncio.

The contents of the registers on entry to the exit are:

## Register

### Contents

**0**

Undefined

**1**

Parameter list address

**2-12**

Undefined

**13**

Address of a 136-byte save area. The first two words are reserved for standard save area conventions, and must not be used.

**14**

Return address

**15**

Entry address

**AR 0-15**

Undefined

10. On return from the exit, the entry environment must be restored.

On return from the exit, the register contents must be:

## Register

### Contents

**2-13**

Restored from the entry values

**0,1,14,15**

Undefined

## AR 0-15

Restored from the entry values

No return of status information is defined for the exit program.

11. You can cancel prior requests to asyncio with the Aio#Cancel function of asyncio.

AioFd is set to the descriptor of the original operation, and:

- To cancel a specific request, AioBuffPtr is set to the address of the Aiocb of the request that is to be canceled.

Your program does not have to worry about timing or serialization with regard to the Aiocb that is being canceled. If that request has already finished, this attempt to cancel it will be ignored.

- To cancel all outstanding asyncio requests on a specific descriptor, AioBuffPtr is set to zero.

**Note:** SelPoll operations cannot be canceled this way. You cancel them by specifying the Aiocb of the SelPoll request.

Cancel releases any blocked requests from their waits, and drives through the I/O completion notifications as it does for a failed request. The original AioRc is set to ECANCELED. Any requests that are not currently blocked are allowed to complete normally. If they attempt to enter a blocked wait they are failed with ECANCELED. Exits that are about to be called are still called normally, and those that are running are not interrupted. Aio#Cancel only "cancels" blocked requests, causing them to fail with ECANCELED.

Synchronous requests (those with AioSync) are also broken out of their blocking waits. They are returned with an AioRc of ECANCELED.

Usually the Aio#Cancel function waits until all I/O completion notifications have finished before asyncio returns to the caller. When asyncio returns, all exits have run, ECBs have been posted, and signals have been sent. The system is finished with the original request Aiocbs and buffers, and they can be freed by the application, subject to its own design. Note that for ECB and signal notifications, there is no coordination with the waiter or signal receiver, so there may still be application code running that is dealing with the request that has just been canceled. Because of timing, you can never tell which requests will finish normally and which requests will be canceled. You know, however, that the system is finished with the request and that any I/O complete notifications that are to be issued have been issued when asyncio returns to your program.

In this case the successful Return\_value is 1, and AioRv contains one of the following values:

- AIO\_CANCELED (1), if the requested operations were canceled.
- AIO\_NOTCANCELED (2), if at least one of the requested operations cannot be canceled because it is in progress.
- AIO\_ALLDONE (3), if all of the operations have already completed; that is, nothing was found to be canceled.

If you do not want to wait for all I/O completion notifications, you can set the AioCancelNoWait flag. In this case, the Return\_value is 0 if any requests were found to be canceled. Your program must maintain its own tally of requests still outstanding if this is significant to it. If no requests were found to be canceled, the Return\_value is 1, and AIO\_ALLDONE is returned in AioRv.

**Note:** A program cannot wait for a cancel operation if it is running on the same TCB that pending exits would run on.

If you do not want the I/O completion notification to be issued, you can set the AioCancelNoNotify flag. If the request is still outstanding at the time of the cancel, the I/O completion notification is suppressed. This means that a specified exit program that has not already run will not be run. Setting this flag also stops the system from updating the Aiocb with the results of the operation, so that the AioRc field tends to remain with a value of EINPROGRESS. TCB exits that were scheduled to run at the time of an I/O completion but that have not yet run when the cancel is issued are skipped.

**Note:** The effectiveness of this flag is unpredictable, because the I/O completion notification may be in progress, or it may already have been made.

Canceling all requests on a given descriptor does not stop new requests from being made, or otherwise affect the descriptor. The program can start afresh or close the descriptor, depending on why it issued the cancel.

The asynchronous features of asyncio do not apply to Aio#Cancel; that is, you cannot specify a signal, an exit program, or an ECB. AioTimeout does not apply to Aio#Cancel.

Cancel succeeds regardless of whether any outstanding requests have been found to cancel.

An individual request can be canceled only once. Subsequent attempts to explicitly cancel the same request fail with EALREADY.

Aio#Cancel cannot be used to cancel any operations other than those that are started with asyncio. You cannot cancel a read(), for example.

A cancel operation itself is not cancelable.

12. Closing a descriptor deletes all requests that are still cancelable on that descriptor. I/O completion notifications are not issued for these requests. If you need exits to be run or ECBs to be posted, you must issue cancel for the descriptor before you issue close for the descriptor.

In most cases, close() will flush out and wait for requests that are still in progress to be deleted. However, it cannot wait for requests that are already in the I/O complete exit programs; or that are just about to call these exits, post the I/O complete ECB, or send the I/O complete signal. Consequently, application code that are related to asyncio requests on the just-closed descriptor may still be in progress when the close() function returns.

Descriptors that are part of an Aio#SelPoll request are removed from that operation. The request remains outstanding, and may complete as a result of activity on one of the other descriptors or when it times out. If all the descriptors for a particular SelPoll happen to be closed, no special action is taken; the request either times out or hangs forever.

13. Not all asynchronous operations support being called to start another I/O before the prior I/O has completed on that same socket. First of all, and most important, each call must have its own Aiocb and buffer or data areas; otherwise a serious and immediate error occurs for all of the operations, and the results are very unpredictable if two operations are using the same areas. In general, starting two or more asynchronous operations on a single socket is analogous to having two or more threads calling the regular synchronous versions of these operations at the same time, and the results are pretty much the same.

Aio#Accept and Aio#ANR may be called more than once. Each inbound connection request will complete a distinct call.

Aio#Connect: Stream (TCP) sockets may not be connected more than once. It does not make sense to connect UDP sockets several times simultaneously, because each connection replaces the previous one, and results will be unpredictable. The results of issuing requests that depend on the connection, such as Aio#Write, before the connection has completed are unpredictable.

Aio#Read, Aio#ReadV, Aio#Recv, Aio#RecvMsg, Aio#RecvFrom: For stream (TCP) sockets, the receive-type operations should not be called more than once before each call completes, as the results are unpredictable. The main reason for this is that the arrival of any data from the network can start the completion of one of these requests while the actual data movement occurs later, and so the data on the stream can be received by different threads out of order.

Aio#Write, Aio#WriteV, Aio#Send, Aio#SendTo, Aio#SendMsg: For stream (TCP) sockets, the send-type operations should not be called more than once before each call completes. Data may be transmitted on the network out of order, and, in general, results are unpredictable. For datagram (UDP) sockets, the send-type operations may be called more than once, because each distinct call defines a single datagram, and there is no implied order of arrival in UDP for these datagrams. Beware of sending too much data, though. If there is network congestion, or the receiver is slow, you can tie up a large amount of system storage with uncontrolled asynchronous sends, and eventually the BPX1AIO calls will start to fail with ENOBUFS.

Aio#SelPoll may be called more than once, but be aware that any one event will complete all the calls at the same time.

Aio#Cancel is not an asynchronous operation.

14. A socket must not be set to nonblocking state if you want I/O completion to wait for data.

If the socket is in nonblocking state and there is no data available, either the asyncio request has its I/O completion driven very quickly with an AioRc of EWOULDBLOCK, or the asyncio call fails with a return code of EWOULDBLOCK.

Note that Aio#ANR does not support nonblocking I/O.

15. Signals do not interrupt asynchronous operations unless they lead to the termination of the caller's process.
16. The Aio#SelPoll command can be used for either an asynchronous select() function or an asynchronous poll() function. The poll() interface structure is used in both cases. AioBuffPtr contains the address of a PollFD array, from BPXYPoll, and AioBuffSize contains the number of elements in the array.
  - For the poll function, the PollFD structure is used in the same way as for poll (BPX1POL, BPX4POL).
  - For the select function, the SelFlags member of the Sel structure from BPXYSEL is mapped over the PollEvents and PollRevents members of the PollFD structure for input and output, respectively. The select event bits have the same meaning as they do for select (BPX1SEL, BPX4SEL), but they are input and output with the technique used by poll events. The triple bit map scheme of select (BPX1SEL, BPX4SEL) is not used.

These bits occupy different bytes in the PollEvents field, and the intended function is determined according to which bits are used. If no bits are set, the operation is considered to be a poll for nothing, rather than a select for nothing.

The entire PollFD array must consistently use only one type of bit. You cannot use select and poll bits for the same file descriptor, nor can you use select bits for one descriptor and poll bits for another. For the sake of performance, the input array is not checked to enforce this rule, and results are unpredictable if the rule is broken. The first occurrence of select bits that are turned on causes the operation to be a select() rather than a poll().

The AioTimeOut field can be used to specify a timeout value for the operation.

Aio#SelPoll can only be used with socket descriptors.

Aio#SelPoll operations cannot be canceled by descriptor; the specific aiocb must be canceled.

Negative descriptors in the PollFd array are ignored, as documented for poll(). Otherwise, the first bad descriptor causes the whole operation to fail at that point in the array. This is a little different from the behavior of poll().

**Guideline:** For performance reasons, do not use asynchronous select or poll if you can use any other asynchronous operation on each descriptor.

For example, doing Aio#Read for each of five sockets is much faster and more efficient than doing one Aio#SelPoll for that same set of sockets. This is because when an Aio#Read completes for one socket you have the data; the other sockets are unaffected and remain ready for inbound data. On the next Aio#Read only that one socket has to be readied again. When Aio#SelPoll completes for any socket, all the others are taken out of their prepared state. You still have to issue another call to actually get the data. On the next Aio#SelPoll all the sockets must be "put back" into their prepared state again.

17. Asynchronous I/O is similar to the select() and poll() functions in that you can wait for data from many different descriptors at the same time. Asynchronous I/O, though, is much faster and much more efficient for large numbers of descriptors. With the asyncio service you also have control over when you wait for the next event.
18. ALETS are generally usable only for synchronous requests (AioSync), with the exception of recvmmsg/sendmsg. A preprocessing exit (AioCallB4) could update the SRB it is running on with an ALET for a data space, but this would add too many instructions to the operation to be practical for the general read or write. You could, however, consider using a Common Area Data Space (CADS).

## Related services

- [“accept \(BPX1ACP, BPX4ACP\) — Accept a connection request from a client socket” on page 13](#)
- [“connect \(BPX1CON, BPX4CON\) — Establish a connection between two sockets” on page 139](#)
- [“poll \(BPX1POL, BPX4POL\) — Monitor activity on file descriptors and message queues” on page 564](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“readv \(BPX1RDV, BPX4RDV\) — Read data and store it in a set of buffers ” on page 673](#)
- [“recv \(BPX1RCV, BPX4RCV\) — Receive data on a socket and store it in a buffer” on page 680](#)
- [“recvfrom \(BPX1RFM, BPX4RFM\) — Receive data from a socket and store it in a buffer” on page 683](#)
- [“recvmsg \(BPX2RMS, BPX4RMS\) — Receive messages on a socket and store them in message buffers ” on page 687](#)
- [“send \(BPX1SND, BPX4SND\) — Send data on a socket ” on page 726](#)
- [“select/selectex \(BPX1SEL, BPX4SEL\) — Select on file descriptors and message queues ” on page 706](#)
- [“sendmsg \(BPX2SMS, BPX4SMS\) — Send messages on a socket ” on page 734](#)
- [“sendto \(BPX1STO, BPX4STO\) — Send data on a socket ” on page 738](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)
- [“writev \(BPX1WRV, BPX4WRV\) — Write data from a set of buffers ” on page 1036](#)
- [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue ” on page 465](#)

## Characteristics and restrictions

The asyncio service supports AF\_INET and AF\_INET6 sockets. It cannot be used with AF\_UNIX sockets.

## Examples

See [“BPX1AIO \(asyncio\) example” on page 1221](#).

# **attach\_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program**

## Function

The attach\_exec callable service attaches a task to run a z/OS UNIX executable program in a newly created child process of the caller. The child process that is created has the same attributes that a child process would have if it were created by the fork service and followed immediately by a call to the exec service. The new process is created in the same address space as the caller, and is a subtask of the caller's task.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1ATX):</b>	31-bit
<b>AMODE (BPX4ATX):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax has the following format:

```
CALL BPX1ATX, (Pathname_length,
               Pathname,
               Argument_count,
               Argument_length_list,
               Argument_list,
               Environment_count,
               Environment_data_length,
               Environment_data_list,
               Exit_routine_address,
               Exit_parameter_list_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ATX with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

**Parameters****Pathname\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the path name of the file. The length can be up to 1023 bytes long.

**Pathname**

Supplied parameter

**Type:**

Character string.

**Character set:**

No restriction.

**Length:**

Specified by the Pathname\_length parameter.

The name of a field that contains the fully qualified path name of the file to be run. Each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

**Argument\_count**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the number of pointers in the lists for the Argument\_length\_list and the Argument\_list parameters. If the program needs no arguments, define Argument\_count as the name of a fullword that contains 0.

**Argument\_length\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an argument to be passed to the specified program. If the program needs no arguments, define `Argument_length_list` as the name of a fullword (doubleword) that contains 0.

**Argument\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an argument to be passed to the specified program. Each argument is of the length that is specified by the corresponding element in the `Argument_length_list`. If the program needs no arguments, define `Argument_list` as the name of a fullword (doubleword) that contains 0.

**Environment\_count**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the number of pointers in the lists for the `Environment_data_length` and the `Environment_data`. If the program needs no environment data, define `Environment_count` as the name of a fullword that contains 0.

**Environment\_data\_length**

Supplied parameter

**Type:**

Structure

**Length:**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an environment variable to be passed to the specified program. If the program does not use environment variables, define `Environment_data_length` as the name of a fullword (doubleword) containing 0.

**Environment\_data\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable, specified by the `Environment_data_length`

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an environment variable to be passed to the specified program. Each environment variable is of the length that is specified by the corresponding element in the `Environment_data_length`. If the program does not use environment variables, define `Environment_data_list` as the name of a fullword (doubleword) that contains 0. If the target executable file is a program that is enabled by



Language Environment, the environment variables that are supplied to this service must include the null terminator as part of the data string and length.

### **Exit\_routine\_address**

Supplied parameter

#### **Type:**

Address

#### **Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be invoked, define `Exit_routine_address` as the name of a fullword (doubleword) that contains 0. Currently the exit must be RMODE 31, and therefore the address must reside below the 2-gigabyte bar.

### **Exit\_parameter\_list\_address**

Supplied parameter

#### **Type:**

Address

#### **Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value that is contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be invoked or does not require parameters, define `Exit_parameter_list_address` as the name of a fullword (doubleword) that contains 0.

### **Return\_value**

Returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword in which the `attach_exec` service returns the process ID of the created child process, if it is successful. If it is not successful, the service returns -1.

### **Return\_code**

Returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword in which the `attach_exec` service stores the return code. The `attach_exec` service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The `attach_exec` service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EAGAIN</b>	The resources required for another process to be created are not available now; or you have already reached the maximum number of processes you are allowed to run. The following reason codes can accompany the return code: JRMaxChild, JRMaxProc, JRMaxUIDs.

Return_code	Explanation
<b>EACCES</b>	The caller does not have appropriate permissions to run the specified file. It may lack permission to search a directory that is named in the Pathname parameter; it might lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type. The following reason code can accompany the return code: JRExecNotRegFile.
<b>EFAULT</b>	A bad address was received as an argument of the call, or the user exit program checked. The following reason codes can accompany the return code: JRExecParmErr and JRExitRtnError.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMVSSAF2ERR</b>	The executable file is a set_user_ID or set_group_ID file, and the file, owner's UID, or GID is not defined to RACF.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or some component of the Pathname is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No Pathname was specified, or one or more of the components of the specified Pathname were not found. The following reason codes can accompany the return code: JRExecNmLenZero and JRQuiescing.
<b>ENOEXEC</b>	The specified file has execute permission, but it is not in the proper format to be a process image. Reason_code contains the loader reason code for the error.
<b>ENOMEM</b>	The new process requires more memory than is permitted by the hardware or the operating system. The following reason codes can accompany the return code: JRExecFileTooBig and JRNoSpace.
<b>ENOTDIR</b>	A directory component of Pathname is not a directory.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the attach\_exec service stores the reason code. The attach\_exec service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see *Reason codes in z/OS UNIX System Services Messages and Codes*. For the ENOEXEC Return\_code, Reason\_code contains the loader reason code for the error:

Reason code	Explanation
<b>X'xxxx0C27'</b>	The target file is not in the correct format to be an executable file.
<b>X'xxxx0C31'</b>	The target file is built at a level that is higher than that supported by the running system.

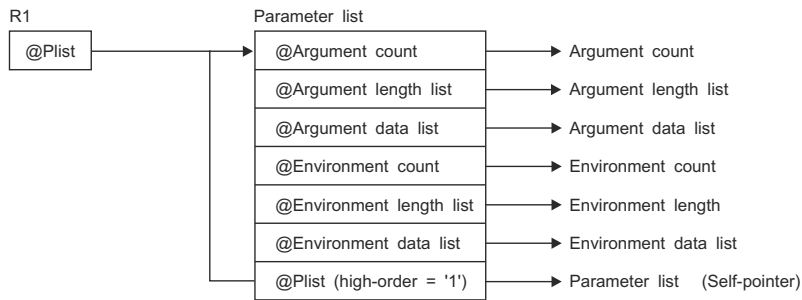
**Usage notes for attach\_exec**

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes attach\_exec to execute a z/OS UNIX executable program file, the program file, and any symbolic links that comprise the path name to the file must have the proper attributes to allow

the invocation. The z/OS UNIX executable file must be installed with an owning UID of 0 or with an owning UID equal to that of the calling set-user-ID program or have the program control extended attribute. The same is true for any symbolic link that comprises the path name to the file or any external symbolic link, except that the program control extended attribute cannot be used because links do not have extended attributes. Additionally, a file or link that is found in a file system mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes.

Not following this setup causes the task that is attached to run the program file to end abnormally with a EC6-xxxxE04B abend, Attach\_exec will return to the caller with a return code of EMVSERR and a reason code of JRecoveryInvoked. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes attach\_exec to run a non-privileged program, the child process that is created to run the non-privileged program inherits the privilege of the set-user-ID or set-group-ID program.

2. The new process (called the *child process*) is similar to the process that calls attach\_exec (called the *parent process*), except for the following differences:
  - The child process has a unique process ID (PID) that does not match any active process group ID.
  - The child has a different parent process ID (namely, the process ID of the process that called attach\_exec).
  - The child has its own copy of the parent's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the parent.
  - If a file has its FCTLCLOFORK or FTCLOEXEC flag set on, it is not inherited by the child process. These flags are set with the fcntl service. For more information, see the fcntl service [“Parameters” on page 217](#).
  - Directories opened via a call to the opendir (BPX1OPD, BPX4OPD) service in the parent process are not inherited by the child process.
  - The process and system utilization times for the child are set to zero.
  - Any file locks previously set by the parent are not inherited by the child.
  - The child process has no interval timers (for example, alarms) set. This is similar to a call to the alarm service with Wait\_time specified as zero.
  - The child process has no pending signals.
  - The child process does not get a copy of the parent's storage, as it would if it were created via a call to the fork service.
  - The child process can address a shared memory segment only while the parent process maintains its attachment.
  - The semaphore adjustment values (semadj) in the child process will be zero.
  - The child process created by this service is terminated when its parent terminates.
3. The executable file to be run receives control with the following attributes:
  - Problem program state.
  - TCB key of caller.
  - AMODE=31(64), taken from the executable.
  - Primary ASC mode.
4. The information that the service passes to the executable file that is to be run is a parameter list pointed to by register 1. The parameter list consists of the parameter addresses listed in this topic. In the last parameter address, the high-order bit is 1.



For AMODE 31 callers, the high-order bit in the last parameter address is 1. For AMODE 64 callers, the high-order bit is part of the 64-bit address. There are always *n* parameters, passed with no end-of-parameter-list indicator.

The last parameter that attach\_exec passed to the executable file identifies the caller of the file as the attach\_exec or exec service.

5. The user exit receives control with the following attributes:

- Problem program state.
- PSW key of caller.
- AMODE=31(64), same as the invoker of BPX1ATX (BPX4ATX).
- Primary ASC mode.

See [“Characteristics and restrictions” on page 57](#) for more information about the execution of the user exit.

6. The register usage on entry to the user exit in AMODE 31 is:

- R0: Undefined.
- R1: Address of the user exit parameter list, as specified by the caller of the attach\_exec service.
- R2–R12: Undefined.
- R13: Address of a 96-byte work area in the same key as the caller of the attach\_exec service.
- R14: The return address from the user exit to the attach\_exec service. This address must be preserved by the user exit.
- R15: Address of the user exit.

7. The register usage on entry to the user exit in AMODE 64 is:

- R0: Undefined.
- R1: 64-bit address of the user exit parameter list, as specified by the caller of the attach\_exec service.
- R2–R12: Undefined.
- R13: 64-bit address of a 96-byte work area in the same key as the caller of the attach\_exec service. Bits 0–32 of this address are 0.
- R14: The 64-bit return address from the user exit to the attach\_exec service. This address must be preserved by the user exit. Bits 0–32 of this address are 0.
- R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, use the BRANCH RELATIVE instructions for branching within the user exit.

8. To support the creation and propagation of a STEPLIB environment to the new process image, attach\_exec allows for the specification of a STEPLIB environment variable. The following table lists the accepted values and the actions that are taken for each value.

Accepted values	Actions that are taken for each value
STEPLIB=NONE	No STEPLIB DD is to be created for the new process image.
STEPLIB=CURRENT	The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the attach_exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
STEPLIB=Dsn1:Dsn2:,...DsnN	<p>The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.</p> <p>The actual name of the DD is not STEPLIB. Instead, it is a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the specified order. The specified data sets must follow standard MVS data set naming conventions. If those data sets do not follow the standard naming conventions, they are ignored. If they do follow the standard naming conventions but the caller does not have the proper security access to a data set, or if the data set is uncataloged or not in load library format, the data set is ignored.</p> <p>The system will also ignore data sets in the following situations:</p> <ul style="list-style-type: none"> <li>• If the system cannot acquire a shared ENQueue on the data set name because another job has ENQueued it exclusively.</li> <li>• If the data set does not exist. That case is similar to but is not strictly equal to the case of uncataloged data sets.</li> </ul> <p>Because the data sets are ignored, the executable file might run without the proper STEPLIB environment. If a data set is in error due to improper security access, an X'913' abend is generated. The installation can suppress the dump for this abend.</p>

If the STEPLIB environment variable is not specified, the default behavior of the attach\_exec service is the same as if STEPLIB=CURRENT were specified.

The ENQueues on an original STEPLIB DD (from submitting the job) will be DEQueued (freed) if this is standard initiator behavior. The system will acquire new ENQueues for the data sets that are named in the STEPLIB variable each time an exec() type function is called.

For information about STEPLIB performance considerations, see [Tuning performance in z/OS UNIX System Services Planning](#).

9. A prior loaded copy of an z/OS UNIX program is reused by this service under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS XCTL service, with the following exceptions:
  - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
  - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the z/OS UNIX program that is found is in storage modifiable by the caller, the prior copy is not reused.
10. If the specified file name resolves to an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order.
  - For an external link, the external name is used only if the name is eight characters or less; otherwise, the caller receives an error from the attach\_exec service.

- For a sticky bit program, the specified file name is used if it is eight characters or less. Otherwise, the program is loaded from the z/OS UNIX file system.

If the attach\_exec caller is running APF-authorized and the specified sticky bit file or link resolves to an MVS program link-edited AC=1 located in an APF-authorized library, the attributes of the sticky bit file or external link must be set up properly to allow this type of invocation.

- For a sticky bit file, it must be installed with an owning UID of 0 or with the APF extended attribute. The owning UID of 0 requirement would also apply to a symbolic link that resolves to the sticky bit file.
- For an external link, it must be installed with an owning UID of 0.

Also, a file with the APF extended attribute is not allowed if found in a file system that is mounted as NOSETUID. If the specified file name represents a symbolic link to a sticky bit file that has the set-user-id attribute, the symbolic link must have an owning UID of 0 or an owning UID equal to that of the sticky bit file. If the sticky bit file has the set-group-id attribute, the symbolic link must have an owning UID of 0 or an owning GID equal to that of the sticky bit file. A file or link that is found in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Failure to follow this setup causes the task that is attached to run the MVS program to end abnormally with a EC6-xxxxC04A abend when the MVS program is invoked via the attach\_exec service.

11. If the calling parent task is in a WLM enclave, the child task is joined to the same WLM enclave. This allows WLM to manage the parent and child as one "business unit of work" entity for system accounting and management purposes.
12. If the target executable program is a Language Environment-enabled program, the environment variables that are supplied to the service must include the null terminator as part of the string and length.
13. If the \_BPX\_PTRACE\_ATTACH environment variable is set to YES, the target executable program is loaded into user-modifiable storage to allow subsequent debugging. Any additional programs loaded into storage during the execution of the target program are also loaded into user-modifiable storage, with the exception of modules loaded from the LPA.
14. If the BPXK\_SIGDANGER environment variable is set to YES, the process will receive a SIGDANGER signal rather than a SIGTERM signal when an OMVS shutdown is initiated.
15. A thread that issues an attach\_exec or attach\_execmvs might receive an A03 abend if any attached children are still running. To avoid the A03 abend, the thread that issued the attach can use waitpid (BPX1WAT) to determine when the attached process has completed. Then the thread can call mvsprocclp (BPX1MPC) to allow time for a full MVS subtask termination to occur after the child process has terminated.
16. If the caller specifies \_BPXK\_DISABLE\_SHLIB=YES, then future loadhfs() and loadhfs\_extended() system calls will ignore the st\_sharelib attribute and load the program into private storage. If the caller specifies NO (the default), then normal system shared library processing takes place.
17. If the caller specifies \_BPXK\_DISABLE\_SHLIB=YES, then future loadhfs() and loadhfs\_extended() system calls ignore the st\_sharelib attribute and load the program into private storage. If the caller specifies NO (the default), then normal system shared library processing takes place. For more information about \_BPXK\_DISABLE\_SHLIB, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).
18. The following restriction exists for the usage of symbolic links by programs that require a RACF clean environment or that run APF-authorized, and are not run in a started task address space:
  - Any symbolic links that comprise the path name to the target executable file must have an owning UID of 0 and not be found in a NOSECURITY file system to allow the invocation.

Otherwise, the task that is attached to run the program file ends abnormally with a EC6-xxxxE04B abend. The attach\_exec callable service returns to the caller with a return code of EMVSERR and a reason code of JRecoveryInvoked.

## Related services

- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“attach\\_execmvs \(BPX1ATM, BPX4ATM\) — Attach an MVS program ” on page 58](#)
- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)

## Characteristics and restrictions

The user exit is given control in the newly created child process on the attached task before the invocation of the specified program. The user exit should not attempt to use any kernel services. Signals cannot be delivered while in the user exit because the attach\_exec service is still in progress and signal delivery is inhibited.

The setuid, setgid, setegid and seteuid services, if started from a process that is created by this service, affect the calling process and any other processes that exist in the address space. However, in a multiprocessing environment when a process that is created by this service attempts to change the security environment, the request is rejected.

If exec or execmvs is started from a process that was created via the attach\_exec service, the initial thread of the process and all of its subtasks is terminated. A new task is attached to run the specified program. This action does not result in the ending of any other tasks in the calling job step. It also does not end other processes in the address space. Because of this behavior, only unauthorized, non-privileged programs are supported on the invocation of exec and execmvs.

Because the z/OS UNIX file system is not an authorized library, the following restrictions apply:

- Executing a program from z/OS UNIX causes the program environment to become uncontrolled, unless the program is identified as program controlled. (That is, unless the ST\_PROGCTL attribute is ON for the z/OS UNIX program file). Running a z/OS UNIX program with the ST\_PROGCTL attribute set to OFF prevents future invocations of authorized programs like Program Access to Data Sets (PADS). These are programs that are given special authorization by the installation and by the installed security product (such as RACF) to read or write to protected data sets. In addition, PADS programs should not attempt to load programs from z/OS UNIX with the ST\_PROGCTL attribute OFF, because these programs are considered uncontrolled and could have been modified by users that do not have the same level of authorization as the PADS program.
- System key, supervisor state, and APF-authorized callers should not attempt to execute a program from z/OS UNIX, unless the executable file has the APF attribute turned on.
- The set-user-ID programs can only be called by processes running with the same effective user ID as the user ID of the executable file.
- The set-group-ID programs can only be called by processes that are running with the same effective group ID as the group ID of the executable file.

Sticky bit programs that are link-edited as APF-authorized can be called only by callers that run APF-authorized.

The newly attached task that is created for the child process does not share user storage subpools 0-127 with the caller.

The following restriction exists for sticky bit programs that have the set-user-ID or set-group-ID attributes:

- If the program is found in the MVS program search order, the MVS program name must have a BPX.STICKYSUG.program\_name resource profile defined in the RACF FACILITY class before it can be successfully attached.

Failure to follow this restriction will cause the caller of attach\_exec to end abnormally with a EC6-xxxxE055 abend.



**Examples**

See “BPX1ATX (attach\_exec) example” on page 1223.

**MVS-related information**

Because the newly created child process runs on a subtask in the same address space as the caller, it has access to the same MVS environment as the caller. The access includes the same allocation (DDs) and storage environment. Programs that run on each of these tasks must be careful not to interfere with other programs that are running in the same environment. Although the child subtask has access to the same storage as the calling task, it does not share any user subpools with the calling task; thus it cannot free any user storage that is obtained by the calling task.

## **attach\_execmvs (BPX1ATM, BPX4ATM) – Attach an MVS program**

---

**Function**

The attach\_execmvs service attaches a task to run an MVS executable program in a newly created child process of the caller. The new process is run in a subtask in the same address space.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1ATM)</b>	31-bit
<b>AMODE (BPX4ATM)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1ATM, (Program_name_length,
               Program_name,
               Argument_length,
               Argument,
               Exit_routine_address,
               Exit_parameter_list_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ATM with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

**Parameters**

**Program\_name\_length**  
Supplied parameter



**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length, in the range of 1 to 8 bytes, of the name of the MVS program.

**Program\_name**

Supplied parameter

**Type**

Character string.

**Character set:**

Conforms to naming conventions for members of MVS PDSs.

**Length**

Specified by the Program\_name\_length parameter.

The name of a field that contains the name of the MVS program to be run. The MVS program name must conform to the naming conventions for members of MVS partitioned data sets (PDSs). The program name is 1 - 8 characters long; the program name is the member name without any qualifiers. The specified Program\_name must be in uppercase.

**Argument\_length**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of the argument that is to be passed to the program. The argument can be 0 - 4096 bytes long.

**Argument**

Supplied parameter

**Type**

Character string

**Length**

Specified by the Argument\_length parameter

The name of a field of length Argument\_length that contains the argument that is to be passed to the MVS program.

**Exit\_routine\_address**

Supplied parameter

**Type**

Address

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be invoked, define Exit\_routine\_address as the name of a fullword (doubleword) that contains 0. The exit must be RMODE 31 and the address must reside below the 2-gigabyte bar.

**Exit\_parameter\_list\_address**

Supplied parameter

**Type**

Address

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be invoked or does not require parameters, define Exit\_parameter\_list\_address as the name of a fullword (doubleword) containing 0.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the attach\_execmvs service returns the process ID of the created child process, if it is successful. If it is not successful, the service returns -1.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the attach\_execmvs service stores the return code. The attach\_execmvs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The attach\_execmvs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>E2BIG</b>	The number of bytes used by the new process image's argument list is greater than the system-imposed limit of 4096 bytes. The following reason code can accompany the return code: JRMVSTooBig.
<b>EFAULT</b>	The user exit program checked. The following reason code can accompany the return code: JRExitRtnError.
<b>ENAMETOOLONG</b>	The specified MVS program name is too long. The length that is specified by Program_name_length is longer than 8 bytes.
<b>ENOENT</b>	The specified MVS program was not found in the link pack area (LPA) or in a link list data set (LNKLST); or the program name argument points to an empty string. The following reason code can accompany the return code: JRExecNmLenZero.
<b>ENOMEM</b>	The new process requires more memory than is permitted by the hardware or the operating system. The following reason codes can accompany the return code: JRExecFileTooBig and JRNoSpace.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the attach\_execmvs service stores the reason code. The attach\_execmvs service returns Reason\_code only if Return\_value is -1. Reason\_code further

qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes for attach\_execmvs

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes attach\_execmvs to run a program, the child process that was created to run the program inherits the privilege of the set-user-ID or set-group-ID program.
2. The result of a call to the attach\_execmvs service is that a subtask is attached to the calling task to run the specified program in a newly created child process. The newly created subtask becomes the initial thread of the newly created child process.
3. The new child process has similarities to the process that calls attach\_execmvs (called the *parent process*), except for the following:

- The child process has a unique process ID (PID) that does not match any active process group ID.
- The child has a different parent process ID, which is the process ID of the process that called attach\_execmvs.
- The child has its own copy of the parent's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the parent.

**Tip:** BPX1ATM only propagates file descriptors, not streams. If the target program is MVS-style, there is no consideration for fd0, fd1, and fd2 being used as the standard streams when the child program starts.

- If a z/OS UNIX file has its FCTLCLFORK or FCTLCLOEXEC flag set on, it is not inherited by the child process. These flags are set with the fcntl service. For more information, see the fcntl service [“Parameters”](#) on page 217.
  - The child has its own copy of the parent's open directory streams. Each open directory stream in the child can share directory stream positioning with the corresponding directory stream of the parent.
  - The process and system utilization times for the child are set to zero.
  - Any file locks previously set by the parent are not inherited by the child.
  - The child process has no interval timers set. This is similar to the results of a call to the alarm service with Wait\_time specified as zero.
  - The child has no pending signals.
  - The child process does not get a copy of the parent's storage, as it would if it were created via a call to the fork service.
  - The child process that is created by this service is terminated when its parent terminates.
4. The input passed to the MVS executable program by the service is consistent with the input passed to MVS programs. On input, the MVS program receives a single-entry parameter list that is pointed to by register 1. The high-order bit of the sole parameter entry is set to 1.

The sole parameter entry is the address of a 2-byte length field followed by an argument string. The length field describes the length of the data that follows it. If a null argument and argument length are specified in the call, the length field specifies 0 bytes on input to the executable program.

5. The MVS program to be run receives control with the following attributes:
  - Problem program state
  - TCB key of caller
  - AMODE=31(64), taken from the executable
  - Primary ASC mode

The specified program can be located in the link pack area (LPA), in a link list data set, job library, step library, or task library. The program search order that is followed is identical to that of the MVS Attach service when the EP parameter is specified.

6. The user exit receives control with the following attributes:
  - Problem program state
  - PSW key of caller
  - AMODE=31(64)
  - Primary ASC mode
7. The register usage on entry to the user exit in AMODE 31 is:
  - R0: Undefined.
  - R1: Address of the user exit parameter list as specified by the caller of the exec service.
  - R2–R12: Undefined
  - R13: Address of a 96-byte work area in the same key as the caller of the exec service.
  - R14: The return address from the user exit to the exec service. This address must be preserved by the user exit.
  - R15: Address of the user exit.
8. The register usage on entry to the user exit in AMODE 64 is:
  - R0: Undefined.
  - R1: 64-bit address of the user exit parameter list as specified by the caller of the exec service.
  - R2–R12: Undefined
  - R13: 64-bit address of a 96-byte work area in the same key as the caller of the exec service. Bits 0–32 of this address are 0.
  - R14: The 64-bit return address from the user exit to the attach\_exec service. This address must be preserved by the user exit. Bits 0–32 of this address are 0.
  - R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.
9. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the attach\_execmvs service are propagated to the new process image. This causes the program that is invoked to run with the same MVS program search order as its invoker.
10. To support the creation and propagation of a STEPLIB environment to the new process image, attach\_execmvs allows for the specification of a STEPLIB environment variable. The following are the accepted values for the STEPLIB environment variable and the actions that are taken for each value:

Value	Action taken
STEPLIB=NONE	No STEPLIB DD is created for the new process image.
STEPLIB=CURRENT	The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
STEPLIB=Dsn1:Dsn2:,...DsnN	<p>The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.</p> <p><b>Note:</b> The actual name of the DD is not STEPLIB. Instead, it is a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order specified. The specified data sets must follow standard MVS data set naming conventions. If they do not follow the standard naming</p>

Value	Action taken
	<p>conventions, they are ignored. The data sets are also ignored if they do follow the standard naming conventions but these situations exist:</p> <ul style="list-style-type: none"> <li>• The caller does not have the proper security access to a data set.</li> <li>• A data set is uncataloged or is not in load library format.</li> </ul> <p>The system will also ignore the data set and continue allocating the other data sets in these situations:</p> <ul style="list-style-type: none"> <li>• If the system cannot acquire a shared ENQueue on the data set name because another job has ENQueued it exclusively.</li> <li>• If the specified data set does not exist. This case is similar but not strictly equal to the case of uncataloged data sets.</li> </ul> <p>Because the data sets are ignored, the executable file might run without the proper STEPLIB environment. If a data set is ignored due to improper security access, an X'913' abend is generated. The dump for this abend can be suppressed by your installation</p> <p>If the program to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For more information about the sanction list, see <a href="#">Using sanction lists in z/OS UNIX System Services Planning</a>. For information about STEPLIB performance considerations, see <a href="#">Controlling use of STEPLIBs in z/OS UNIX System Services Planning</a>.</p>

If the STEPLIB environment variable is not specified, the default behavior of the attach\_execmvs service is the same as if STEPLIB=CURRENT were specified.

If the program to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For more information about the sanction list, see [Using sanction lists in z/OS UNIX System Services Planning](#). For information about STEPLIB performance considerations, see [Tuning performance in z/OS UNIX System Services Planning](#).

11. If the calling parent task is in a WLM enclave, the child task is joined to the same WLM enclave. After the child task is joined to the enclave, WLM can manage the parent and child as one business unit of work entity for system accounting and management purposes.
12. A thread that issues an attach\_exec or attach\_execmvs might receive an A03 abend if any attached children are still running. To avoid the A03 abend, the thread that issued the attach can use waitpid (BPX1WAT) to determine when the attached process has completed, then call mvspccplp (BPX1MPC) to allow time for a full MVS subtask termination to occur after the child process has terminated.

## Related services

- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“execmvs \(BPX1EXM, BPX4EXM\) — Run an MVS program ” on page 175](#)
- [“attach\\_exec \(BPX1ATX, BPX4ATX\) — Attach a z/OS UNIX program ” on page 48](#)

- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)

## Characteristics and restrictions

The user exit is given control in the newly created child process on the attached task before the invocation of the specified program. This exit can be used by the caller to alter the environment of the child process, similarly to the way in which a program would alter the child's environment after a call to fork, but before the call to execmvs. The user exit should not attempt to use any kernel services from the exit. Signals cannot be delivered while in the user exit, because the attach\_execmvs service is still in progress and signal delivery is inhibited.

The setuid, setgid, setegid and seteuid services, if invoked from a process that is created by this service, affect the calling process and any other processes that exist in the address space.

If exec or execmvs is invoked from a process that was created via the attach\_execmvs service, the initial thread task of the process and all of its subtasks are terminated, and a new task is attached to run the specified program. The initial thread task in such a process is the task that was created as a result of the call to the attach\_execmvs service. The call to exec or execmvs does not result in the ending of any other tasks in the calling job step, nor does it end other processes in the address space. Because of this behavior, only unauthorized, non-privileged programs are supported on the invocation of exec and execmvs.

APF-authorized programs can be invoked from this service if the caller is APF-authorized.

## Examples

See [“BPX1ATM \(attach\\_execmvs\) example” on page 1222](#).

## MVS-related information

Because the newly created child process runs on a subtask in the same address space as the caller, it has access to the same MVS environment as the caller. This includes the same allocation (DDs) and storage environment. Because of this, programs that run on each of these tasks should be careful not to interfere with other programs running in the same environment. Although the child subtask has access to the same storage as the calling task, it does not share any user subpools with the calling task. For this reason, it cannot free user storage that is obtained by the calling task.

# auth\_check\_resource\_np (BPX1ACK, BPX4ACK) — Determine a user's access to a RACF-protected resource

---

## Function

The auth\_check\_resource\_np service checks the authority of a RACF-defined user to access a RACF-defined resource. Resources in the DATASET class cannot be checked. The authorization required to invoke this service is one of the following:

- Read access to the BPX.SERVER resource in the FACILITY class.
- A UID of 0 when the BPX.SERVER resource is not defined in the FACILITY class.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1ACK):</b>	31-bit

Operation	Environment
<b>AMODE (BPX4ACK):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax has the following format:

```
CALL BPX1ACK, (Cell_UUID,
               Principal_UUID,
               Userid_Length,
               Userid,
               Security_Class_Length,
               Security_Class,
               Entity_Name_Length,
               Entity_Name,
               Access_Type,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ACK with the same parameters.

## Parameters

### Cell\_UUID

Supplied parameter

**Type:**

Character string

**Length:**

36 bytes

The name of a 36-byte area that contains the cell DCE UUID. If the cell DCE UUID is not specified, the first byte of this 36-byte area must contain NUL (X'00').

### Principal\_UUID

Supplied parameter

**Type:**

Character string

**Length:**

36 bytes

The name of a 36-byte area that contains the principal DCE UUID. If the principal DCE UUID is not specified, the first byte of this 36-byte area must contain NUL (X'00').

### Userid\_Length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Userid parameter. Userid\_Length can be in the range of 0 to 8. If a user ID is not required, specify the name of a fullword that contains zero.

**Userid**

Supplied parameter

**Type:**

Character string

**Character set:**

The XPG4 portable character set, which includes uppercase and lowercase letters (A-Z,a-z), numerics (0-9), period (.), dash (-) and underscore(\_). In addition, the special characters \$, %, and # can be specified. (Since these characters are not part of the XPG4 portable character set, however, you should consider the future possibility of program portability before using these characters.)

**Length:**

Specified by the Userid\_Length parameter

The name of an area, 0 to 8 characters in length, that contains a user ID. If a user ID is not required (Userid\_Length is zero), this parameter is ignored.

**Security\_Class\_Length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Security\_Class. The Security\_Class\_Length must be in the range of 1 to 8.

**Security\_Class**

Supplied parameter

**Type:**

Character string

**Character set:**

Uppercase alphanumeric

**Length:**

Specified by the Security\_Class\_Length parameter

The name of an area, 1 to 8 characters in length, that contains the Security\_Class. The Security\_Class parameter cannot specify DATASET. For systems using RACF, the class name specified must be in the RACF class descriptor table.

**Entity\_Name\_Length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Entity\_Name. The Entity\_Name\_Length can be in the range of 1 to 246.

**Entity\_Name**

Supplied parameter

**Type:**

Character string

**Character set:**

Uppercase alphanumeric

**Length:**

Specified by the Entity\_Name\_Length parameter



The name of an area, 1 to 246 characters in length, that contains the Entity\_Name.

### Access\_Type

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a numeric value that identifies the type of access to check for. The following Access\_Type constants are defined by the BPXYCONS macro. See [“BPXYCONS – Constants used by services” on page 1062](#).

Constant	Access
<b>ACK_READ#</b>	check READ authority
<b>ACK_UPDATE#</b>	check UPDATE authority
<b>ACK_CONTROL#</b>	check CONTROL authority
<b>ACK_ALTER#</b>	check ALTER authority

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the auth\_check\_resource\_np service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the auth\_check\_resource\_np service stores the return code. The auth\_check\_resource\_np service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The auth\_check\_resource\_np service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	<p>One or more of the following conditions was detected:</p> <ul style="list-style-type: none"> <li>• Access_Type specified is undefined</li> <li>• Userid_Length is outside allowable range (0-8)</li> <li>• Security_Class_Length is outside allowable range (1-8)</li> <li>• Entity_Name_Length is outside allowable range (1-246)</li> </ul> <p>The following reason codes can accompany the return code: JRAccessUndefined, JRUserNameLenError, JRClassLenErr, or JREntityLenErr.</p>

Return_code	Explanation
<b>ESRCH</b>	<p>One or more of the following conditions was detected:</p> <ul style="list-style-type: none"> <li>• The user ID is not defined to the security product</li> <li>• No mapping to a user ID exists for the specified UUIDs</li> <li>• The resource is not defined to the security product</li> <li>• The DCEUUIDS class is not active</li> </ul> <p>The following reason codes can accompany the return code: JRSAFNoUser, JRSAFNoUUIDtoUser, JRSAFResourceUndefined, or JRSAFNoDCEClass.</p>
<b>ENOSYS</b>	<p>One or more of the following conditions was detected:</p> <ul style="list-style-type: none"> <li>• No security product is installed</li> <li>• SAF support for this function is not installed</li> </ul> <p>The following reason codes can accompany the return code: JRNoSecurityProduct, or JRNoSAFSupport.</p>
<b>EMVSSAF2ERR</b>	<p>An error occurred in the security product. One or more of the following conditions were detected:</p> <ul style="list-style-type: none"> <li>• An internal error occurred in the security product</li> <li>• An error was detected in the parameter list</li> <li>• There was an undefined return code or reason code</li> </ul> <p>The following reason codes can accompany the return code: JRSAFInternal, JRSAFParmListErr, or JRUnexpectedError.</p>
<b>EPERM</b>	<p>One or more of the following conditions were detected:</p> <ul style="list-style-type: none"> <li>• If BPX.SERVER is defined, the caller does not have update permission to BPX.SERVER. If BPX.SERVER is not defined, the caller is not a superuser.</li> <li>• The user does not have the access specified to the resource.</li> <li>• The caller's address space has done a load from an uncontrolled library</li> </ul> <p>The following reason codes can accompany the return code: JRNotServerAuthorized, JRNoResourceAccess, or JREnvDirty.</p>
<b>Reason_code</b>	
Returned parameter	
<b>Type:</b>	
Integer	
<b>Length:</b>	
Fullword	
<p>The name of a fullword in which the auth_check_resource_np service stores the reason code. The auth_check_resource_np service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For a list of reason codes, see <a href="#">Reason codes in z/OS UNIX System Services Messages and Codes</a>.</p>	

## Usage notes

1. The ability to query a user's access to protected resources is a privileged operation. An installation has the following ways of allowing an application to use this service:
  - For the highest level of security, the installation can define the BPX.SERVER resource in the FACILITY class. In order for the application to access this service, it must have at least read access to this

profile. In addition, all load modules executing in the application's address space must be defined to RACF. For more information about setting up this security, see [Establishing UNIX security in z/OS UNIX System Services Planning](#).

- For a lower security arrangement, assign a UID of 0 to the user ID with which the application is run, so that it operates as a superuser.
2. This service cannot be used to determine access to POSIX resources.
  3. The access check can be made with several forms of identity. The first identity that is specified in the following list is used to make the authorization check:
    - a. User ID.
    - b. Principal/Cell UUIDs.
    - c. Caller's task level ACEE.
    - d. Caller's address space level ACEE.
  4. When no identity is specified by the caller and the caller's task has an ACEE created with pthread\_security\_np (BPX1TLS, BPX4TLS) for a SURROGATE (non-password) client, both the task and address space level ACEEs are used in determining the type of access permitted to a resource.
  5. Both the principal and cell UUIDs are in string form. A UUID string is 36 characters long. The string must contain the delimiter - in character positions 9, 14, 19, and 24. The general form of a UUID string is xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, where x represents a valid numeric or hexadecimal character.

## Related services

- [“pthread\\_security\\_np, pthread\\_security\\_applid\\_np \(BPX1TLS, BPX4TLS\) — Create or delete thread-level security” on page 601](#)

## Characteristics and restrictions

The auth\_check\_resource\_np service is restricted to users that have the appropriate privileges.

## Examples

See [“BPX1ACK \(auth\\_check\\_resource\\_np\) example” on page 1220](#).

# **\_\_authenticate (BPX1AUT, BPX4AUT) — Authenticate the specified user's credentials**

---

## Function

The \_\_authenticate() callable service can authenticate a user by using passwords, password phrases, PassTickets, or Identity Tokens (IDTs). It can also optionally generate an IDT to be used for follow-on authentications.

**Restriction:** The \_\_authenticate() service authenticates only the user's credentials. It does not create a security context (ACEE) or modify the caller's process or thread identity.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1AUT)</b>	31-bit

Operation	Environment
<b>AMODE (BPX4AUT)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1AUT(Auth_cred_type,  
             User_name_length,  
             User_name,  
             Pass_length,  
             Pass,  
             New_pass_length,  
             New_pass,  
             Idt_buffer_length,  
             Idt_buffer_ptr,  
             ltd_length,  
             Msg_buffer_subpool,  
             Msg_buffer_ptr,  
             Appl_id_length,  
             Appl_id,  
             Option_flags,  
             Return_value,  
             Return_code,  
             Reason_code)
```

AMODE 64 callers use BPX4AUT with the same parameters.

## Parameters

### Auth\_cred\_type

Supplied parameter.

The name of a fullword that identifies the type of credentials to be used to perform the authentication.

#### Constant

##### Description

#### AUTH\_USER\_ID#

The User\_name parameter specified by the caller is passed to RACF for the authentication.

#### AUTH\_ID\_TOKEN#

The IDT from the location specified by the Idt\_buffer\_ptr parameter is passed to RACF for authentication.

Both AUTH\_USER\_ID# and AUTH\_ID\_TOKEN# can be specified together. When both are specified, both the user ID and IDT are passed to RACF. If the user ID associated with the IDT does not match the User\_name parameter, the authentication will fail.

### User\_name\_length

Parameter supplied and returned.

#### Type

Integer.

#### Length

Fullword.

Supplied parameter when AUTH\_USER\_ID# is specified for the Auth\_cred\_type parameter. It specifies the length of the User\_name parameter string.

Returned length of the User\_name parameter string that is returned when authenticating with only an IDT. The name is obtained from a temporary ACEE created by RACF. The caller must specify the maximum user ID length of 8 characters to accommodate all possible user ID sizes. The authenticate() service will modify the User\_name\_length parameter specified by the caller to indicate the length of the returned User\_name parameter string.

**User\_name**

Parameter supplied and returned.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the User\_name\_length parameter.

The name of a field where the user ID is supplied or returned.

Supplied user ID when Auth\_cred\_type specifies AUTH\_USER\_ID#.

Returned user ID when Auth\_cred\_type specifies AUTH\_ID\_TOKEN# and AUTH\_USER\_ID# is not specified

**Pass\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket. For a password phrase, it must be between 9 and 100 characters. A length of zero indicates that Pass is to be ignored.

**Pass**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the Pass\_length parameter.

The left-aligned name of a field of length Pass\_length that contains the password, PassTicket, or password phrase that is to be verified.

**New\_pass\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the length of New\_pass. This length must be between 1 and 8 characters for a password. For a password phrase, it must be between 9 and 100 characters. A length of zero indicates that New\_pass is to be ignored.

**New\_pass**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the New\_pass\_length parameter.

The left-aligned name of a field of length New\_pass\_length that contains the new password or password phrase.

**Idt\_buffer\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

Supplied length of the buffer pointed to by Idt\_buffer\_ptr. If the supplied buffer is smaller than the IDT built by RACF, the syscall fails with -1, EINVAL, JrbuffTooSmall, and the required length is returned in the Idt\_length parameter. In this situation, the caller should allocate a larger buffer and retry the \_\_authenticate() call.

**Idt\_buffer\_ptr**

Parameter supplied and returned.

**Type**

Address.

**Length**

Fullword (doubleword).

Supplied address of the buffer that can be used for input or output of an IDT.

When AUTH\_ID\_TOKEN# is specified for the Auth\_cred\_type parameter, the buffer contains an IDT that the \_\_authenticate() service will pass to RACF to be authenticated.

When only AUTH\_USER\_ID# is specified for the Auth\_cred\_type parameter and AUTH\_BUILD\_IDT# is specified for the Option\_flags parameter. If an IDT is generated, the \_\_authenticate() service copies the newly generated IDT into the buffer.

In both cases, if an IDT is copied into the buffer, the #AUTH\_RETURNED\_IDT# flag in the Option\_Flags parameter is set by \_\_authenticate() and returned to the caller.

**Idt\_length**

Parameter supplied and returned.

**Type**

Integer.

**Length**

Fullword.

Supplied length of the IDT whose location is specified by the Idt\_buffer\_ptr parameter when:

- AUTH\_ID\_TOKEN# is specified for the Auth\_cred\_type parameter or
- Must be zero when AUTH\_USER\_ID# is specified for the Auth\_cred\_type parameter and AUTH\_BUILD\_IDT# is specified for the Option\_flags parameter.

Returned length of the IDT copied into the location that is specified by the Idt\_buffer\_ptr parameter when:

- AUTH\_ID\_TOKEN# is specified for the Auth\_cred\_type parameter and RACF refreshes the IDT while authenticating the IDT supplied by the caller.

- AUTH\_USER\_ID# is specified for the Auth\_cred\_type parameter and AUTH\_BUILD\_IDT# is specified for the Option\_flags parameter.

The Idt\_length parameter can be both supplied and returned for the same syscall depending on the conditions specified earlier being met or not.

**Msg\_buffer\_subpool**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword field that contains the storage subpool number to be used by RACF to obtain the message buffer. The Msg\_buffer\_subpool must be in the range 0 to 127. If RACF obtains a message buffer, the address of the buffer is returned to the caller in the Msg\_buffer\_ptr parameter.

**Msg\_buffer\_ptr**

Returned parameter.

**Type**

Address.

**Length**

Fullword (doubleword).

The name of a fullword (doubleword) field in which the address of a message buffer obtained by RACF is returned to the caller. The Msg\_buffer\_ptr parameter will only be returned when RACF obtains a message buffer. The caller is responsible to free the buffer storage. The AUTH\_MSGRTRN# option in the Option\_Flags parameter must be specified to have messages returned.

**Appl\_id\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword field that contains the length of the Appl\_id parameter. This length must be between 1 and 8 characters. A length of zero indicates that the Appl\_id parameter is to be ignored.

**Appl\_id**

Supplied parameter.

**Type**

Character string.

**Length**

Specified by the Appl\_id\_length parameter.

**Character set**

No restriction.

The left-aligned name of a field of length Appl\_id\_length containing the APPLID that identifies the name of the application requesting authentication. If an Appl\_id is not specified (Appl\_id\_length of zero), the application ID will default to OMVSAPPL.

**Option\_flags**

Parameter supplied and returned.

**Type**

Structure.

**Length**

Fullword.

## **\_\_authenticate (BPX1AUT, BPX4AUT)**

The name of a fullword binary field that contains the BPX1AUT/BPX4AUT options. If no options are required, specify the name of a fullword that contains 0.

Valid values for this field include the following:

### **AUTH\_BUILD\_IDT#**

Request that an ID Token be built by RACF and returned to the caller. This option is valid when only AUTH\_USER\_ID# is specified (AUTH\_ID\_TOKEN# is not specified) for the Auth\_cred\_type parameter. The newly built ID Token is returned to the caller in the buffer pointed to by the Idt\_buffer\_ptr parameter. Its length is returned in the Idt\_length parameter.

### **AUTH\_RETURN\_USERNAME#**

Request that the user ID that is associated with the ID Token used for authentication be returned in the field that is specified by the User\_name parameter. This option is valid when only AUTH\_ID\_TOKEN# is specified (AUTH\_USER\_ID# is not specified) for the Auth\_cred\_type parameter and the User\_name\_length parameter contains a value of 8. If those conditions are not met, the request will fail. The length of the returned User\_name is returned in the User\_name\_length parameter.

### **AUTH\_MSGRTRN#**

This option controls the MSGRTRN parameter specified on the RACROUTE REQUEST=VERIFY call made by \_\_authenticate().

When this option is not specified, MSGRTRN=NO (default) is specified for the resulting RACROUTE REQUEST=VERIFY. Messages are not returned in a buffer and are issued by RACF using TPUT.

When this option is specified, MSGRTRN=YES is specified for RACROUTE REQUEST=VERIFY. The address of the message buffer obtained by RACF is returned in the Msg\_buffer\_ptr parameter. The buffer is allocated by RACF in the subpool specified by the Msg\_buffer\_subpool parameter. The caller is responsible to free the returned message buffer.

### **AUTH\_RETURNED\_IDT#**

This option is returned by the \_\_authenticate system call to indicate an IDT has been copied into the location specified by the Idt\_buffer\_ptr parameter. The length of the IDT is returned in the Idt\_length parameter. This can occur when the caller requested to build a new IDT or is authenticating with an IDT and RACF refreshes the IDT in response to changes of system or user settings. This option should not be specified by the caller. If it is, the system call fails and gives a return value of -1 with a return code of EINVAL and a reason code of J1BadOptnFlags.

## **Return\_value**

Returned parameter.

### **Type**

Integer

### **Length**

Fullword

The name of a fullword in which the \_\_authenticate service returns -1 if it is not successful. If it is successful, the \_\_authenticate service returns a value of zero.

## **Return\_code**

Returned parameter.

### **Type**

Integer

### **Length**

Fullword

The name of a fullword in which the \_\_authenticate service stores the return code. The \_\_authenticate service returns the Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The \_\_authenticate service can return one of the following values in the Return\_code parameter:



<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	Permission is denied.
<b>EINVAL</b>	The parameter is incorrect. The following reason codes can accompany the return code: JrBuffTooSmall, JrInvalidParms, JrUserNameLenError, JrPasswordLenError, JrNewPasswordLenError, JrUserNameBad, JrBadOptnFlags, JrAuthBadMsgBuffSubpool, JrAuthCredTypeInvalid, and JrIncompatOptnFlags.
<b>ESRCH</b>	No such process or thread exists. The following reason codes can accompany the return code: JrUserIdUndefToRacf.
<b>EMVSSAFEXTERR</b>	A RACF extract error has occurred.
<b>EMVSSAF2ERR</b>	A RACF error has occurred. The following reason codes can accompany the return code: JrICSFNotActive, JrICSFUnknownErr, JrAuthIDTInvalid, JrAuthUserIDMismatch, JrIDTUnsigned, JrIDTApplMismatch, and JrNoIDTGenerated.
<b>EMVSEXPIRE</b>	The password or identity token for the specified resource has expired.
<b>EMVSPASSWORD</b>	The new password or password phrase specified is not valid.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the `__authenticate` service stores the reason code. The `__authenticate` service returns the Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If a profile is defined in the FACILITY class protecting the BPX.DAEMON resource, all programs that are loaded into the caller's address space must be controlled programs by the installed security product (such as RACF). If the `__authenticate` service detects that a load of a non-program control program was done, it fails with an errno of EMVSERR and an errnojr of JRENVDIRTY. For more information, see [Establishing the correct level of security for daemons](#) in *z/OS UNIX System Services Planning*.
2. To request RACF to return messages in a buffer, the AUTH\_MSGRTRN# option in the Option\_flags parameter must be specified in conjunction with the Msg\_buffer\_subpool and Msg\_buffer\_ptr parameters. If the AUTH\_MSGRTRN# option is not specified, the Msg\_buffer\_subpool and Msg\_buffer\_ptr parameters are ignored and the `__authenticate()` service will not request RACF to return messages resulting from the request. If it is not returning messages, RACF will use TPUT to issue the messages. For more information about messages returned by RACF, see the MSGSP and MSGRTRN parameters in [RACROUTE \(standard form\)](#) in *z/OS Security Server RACROUTE Macro Reference*.
3. When a message buffer address is returned to the caller in the Msg\_buff\_ptr parameter, the caller is responsible to free the storage it points to. The format of the buffer is defined by RACF. The area consists of two fullwords followed by the message in write-to-operator (WTO) parameter list format. The first word is the length of the area including the two-fullword header; the second word points to the next message area, if there is one, or contains zero if no more message areas exist. For

more information about messages returned by RACF, see the MSGSP and MSGRTRN parameters in [RACROUTE \(standard form\) in z/OS Security Server RACROUTE Macro Reference](#).

4. The current minimum size of an Identity Token (IDT) is 1024 bytes. The size might change in the future as RACF adds support for more types of IDTs. For more information about IDTs, see [Activating and using the IDTA parameter in RACROUTE REQUEST=VERIFY and initACEE in z/OS Security Server RACROUTE Macro Reference](#).
5. The `__authenticate` service only accepts and returns signed IDTs. For more information about signed and unsigned IDTs, see [Activating and using the IDTA parameter in RACROUTE REQUEST=VERIFY and initACEE in z/OS Security Server RACROUTE Macro Reference](#).
6. The `Idt_buffer_length` and `Idt_buffer_ptr` parameters detail the length and location of the buffer supplied by the caller for storing IDTs. The `Idt_length` parameter details the length of an IDT located in the IDT buffer. The caller specifies the `Idt_length` when supplying an IDT for authentication. The system returns the `Idt_length` when either building a new IDT or for an IDT that has been refreshed by RACF when authenticating with an IDT. When the system returns a new or refreshed IDT, the `#AUTH_RETURNED_IDT#` flag in the `Option_flags` parameter is set and returned to the caller to indicate that a newly created or refreshed IDT was returned.

## Related services

- “`__passwd, __passwd __applid` (BPX1PWD, BPX4PWD) — Verify or change security information” on [page 530](#)

## Characteristics and restrictions

None.

## Examples

“BPX1AUT (`__authenticate`) example” on [page 1223](#) and “BPX4AUT (`__authenticate`) example” on [page 1317](#).

# bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor

---

## Function

The bind callable service binds a unique local name to a socket descriptor.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1BND)</b>	31-bit task or SRB mode
<b>AMODE (BPX4BND)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1BND,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4BND with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the bind is to be done.

### Sockaddr\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the length of Sockaddr.

### Sockaddr

Supplied parameter

#### Type:

Character

#### Length:

Length specified by Sockaddr\_length.

The name of a field that contains the name to be bound to the socket descriptor. The format of Sockaddr is determined by the domain in which the socket descriptor was created. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for additional information on the format of Sockaddr.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the bind service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the bind service stores the return code. The bind service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The bind service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EADDRINUSE</b>	The specified address is already in use. The following reason code can accompany the return code: <code>JRNameExists</code> .
<b>EAFNOSUPPORT</b>	The address family specified in the address structure is not supported.
<b>EBADF</b>	The socket descriptor is incorrect. The following reason codes can accompany the return code: <code>JRFileDesNotInUse</code> , <code>JRFileNotOpen</code> .
<b>EINVAL</b>	One of the input parameters was not valid. The following reason codes can accompany the return code: <code>JRSocketCallParmError</code> , <code>JRSockNoname</code> .
<b>EIO</b>	There has been a network or transport failure. The following reason code can accompany the return code: <code>JRPrevSockError</code> .
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: <code>JROutofSocketCells</code> .
<b>ENOTSOCK</b>	<code>Socket_descriptor</code> does not refer to a valid socket descriptor. The following reason code can accompany the return code: <code>JRMustBeSocket</code> .
<b>EPERM</b>	The user is not permitted to bind to the specified port. The following reason code can accompany the return code: <code>JRUserNotPrivileged</code> .

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the bind service stores the reason code. The bind service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. An application can retrieve the assigned socket name with the `getsockname` service.
2. Sockets in the `AF_UNIX` domain create a name in the file system that must be deleted by the application (using `unlink`) when it is no longer needed.
3. For `Sockaddr` to be returned on an accept request for an `AF_UNIX` domain socket, the client application doing the connect must bind a unique local `Sockaddr` to the socket with the bind request before issuing the connect request.
4. Server applications issue the bind request to register their addresses with the system. Both connection and connectionless servers must do this before accepting requests from clients.
5. For network sockets, the user must have appropriate privileges (see [“Authorization” on page 8](#)) to bind to a port in the range from 1 to 1023.
6. See Chapter 3, [“Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

## Related services

- [“accept \(BPX1ACP, BPX4ACP\) — Accept a connection request from a client socket” on page 13](#)
- [“listen \(BPX1LSN, BPX4LSN\) — Prepare a server socket to queue incoming connection requests from clients” on page 391](#)
- [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets” on page 865](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1BND \(bind\) example” on page 1224](#).

# bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address

---

## Function

The bind with source address selection callable service binds the best source address for the provided destination IP address to an AF\_INET6 socket descriptor.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1BAS):</b>	31-bit task or SRB mode
<b>AMODE (BPX4BAS):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1BAS,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4BAS with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

**bind2addrsel (BPX1BAS, BPX4BAS)**

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword that contains the AF\_INET6 socket file descriptor for which the bind with source address selection is to be done.

**Sockaddr\_length**  
Supplied parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a field that contains the length of Sockaddr.

**Sockaddr**  
Supplied parameter

**Type:**  
Character

**Length:**  
Length specified by Sockaddr\_length.

The name of a field that contains the destination IP address. The best source address for the provided destination address will be selected and bound to the AF\_INET6 socket descriptor.

**Return\_value**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the bind with source address selection service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the bind with source address selection service stores the return code. The bind with source address selection service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The bind with source address selection service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EAFNOSUPPORT	The address family specified in the address structure is not supported.
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.

Return_code	Explanation
<b>EINVAL</b>	One of the input parameters was not valid. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoname, JrInAddrAnyNotAllowed.
<b>EIO</b>	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EOPNOTSUPP</b>	The socket domain type is not supported. The following reason code can accompany the return code: JrIncorrectSocketType.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the bind with source address selection service stores the reason code. The bind with source address selection service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. An application can retrieve the assigned socket name with the getsockname service.
2. An application's suggestion of best source address is set by the IPv6 setsockopt() call socket option SOCK#IPv6\_ADDR\_PREFERENCES.
3. In CINET environment, bind with source address selection service is routed to the TCPIP stack that is most appropriate for the provided destination IP address.
4. For more information about programming considerations for SRB mode, see [Chapter 3, "Callable services for SRB mode routines," on page 1041](#).

**Related services**

- ["bind \(BPX1BND, BPX4BND\) — Bind a unique local name to a socket descriptor" on page 76](#)
- ["accept \(BPX1ACP, BPX4ACP\) — Accept a connection request from a client socket" on page 13](#)
- ["listen \(BPX1LSN, BPX4LSN\) — Prepare a server socket to queue incoming connection requests from clients" on page 391](#)
- ["socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets" on page 865](#)

**Characteristics and restrictions**

1. The bind with source address selection service is for IPv6 and IPv4-mapped IPv6 addresses only.
2. Only TCP and UDP sockets are supported. RAW sockets are not supported.
3. Port number is irrelevant for bind with source address selection service and will not be validated.
4. Destination IP address of IN6ADDR\_ANY is not supported. The bind with source address selection service will fail with EINVAL return code and JrInAddrAnyNotAllowed reason code.

**Examples**

See [“BPX1BAS \(bind with source address selection\) example”](#) on page 1224.

## chattr (BPX1CHR, BPX4CHR) – Change the attributes of a file or directory

---

**Function**

The chattr service modifies the attributes that are associated with a file. It can be used to change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, file format and size, and file tag. It can also be used to set the initial security label for a file or directory. Identify the file by its path name.

For the corresponding service that uses a file descriptor, see [“fchattr \(BPX1FCR, BPX4FCR\) – Change the attributes of a file or directory by descriptor”](#) on page 190.

**Requirements**

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CHR)</b>	31-bit
<b>AMODE (BPX4CHR)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1CHR, (Pathname_length,
               Pathname,
               Attributes_length,
               Attributes,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CHR with the same parameters.

**Parameters*****Pathname\_length***

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.



The name of a fullword that contains the length of the path name of the file whose attributes that you want to change.

**Pathname**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. The length of this field is specified in Pathname\_length.

**Attributes\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the length of the area containing the attributes that you want to change.

**Attributes**

Supplied parameter.

**Type**

Structure.

**Length**

Specified by the Attributes\_length parameter.

The name of the area that contains the attributes that you want to change. The area is mapped by BPXYATT. For more information about the content of this area, see [“BPXYATT — Map file attributes for chattr and fchattr”](#) on page 1057.

For extended attributes, the area is mapped by the attx structure in the same BPXYATT mapping. Storage for the attribute value must be contiguous with the attx structure and the attributes\_length argument is the length of the attx structure plus the length of the attribute value.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the chattr service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the battr service stores the return code. The battr service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The battr service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process did not have appropriate permissions. Possible reasons include: <ul style="list-style-type: none"> <li>The calling process was attempting to set access time or modification time to current time, but the effective UID of the calling process does not match the owner of the file. The process does not have write permission for the file or the process does not have appropriate privileges. (See <a href="#">“Authorization”</a> on page 8.)</li> <li>The calling process was attempting to truncate the file, and it does not have write permission for the file.</li> </ul>
<b>EBUSY</b>	The file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
<b>EEXIST</b>	A request to create an extended attribute was specified, but the attribute is already set. The following reason code can accompany the return code: JrAttxExists.
<b>EFBIG</b>	The calling process was attempting to change the size of a file, but the specified length is greater than the maximum file size limit for the process. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRWriteBeyondLimit.
<b>EINVAL</b>	The extended attribute operation is invalid due to the values that are being passed. The input length for the ATTX parameter might be insufficient or the subcommand AttxSubCmd is invalid. The following reason codes can accompany the return code: JRInvMinArgLenAttx, JRInvArgLenAttx, JRInvAttxCmd.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMVSERR</b>	An MVS environmental error was detected. The following reason code can accompany the return code: JrSeclabelClassInactive.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters. (File name truncation is not supported.)
<b>ENOATTR</b>	For the replace or remove function, the specified extended attribute cannot be set. ENOATTR is also set if the passed in attribute name, AttxName, is empty. The following reason codes can accompany the return code: JrAttxName and JrAttxUnavail.
<b>ENOENT</b>	No file named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOSYS</b>	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.
<b>ENOTDIR</b>	Some component of <i>pathname</i> is not a directory.
<b>ENOTSUP</b>	The request against an extended attribute is not supported. The following reason codes can accompany the return code: JRInvAttxFType, JrSetAttxTrustClas, JRAttxNoSup, and JRSetAttxNameSpc.

**Return\_code****Explanation****EPERM**

The operation is not permitted for one of the following reasons:

- The calling process was attempting to change the mode or the file format. However, the effective UID of the calling process does not match the owner of the file and the calling process does not have appropriate privileges. For more information, see [“Authorization” on page 8](#).
- The calling process was attempting to change the owner, but it does not have appropriate privileges.
- The calling process was attempting to change the general attribute bits, but it does not have write permission for the file.
- The calling process was attempting to set a time value (not current time). However, the effective user ID does not match the owner of the file, and it does not have appropriate privileges.
- The calling process was attempting to set the change time or reference time to current time, but it does not have write permission for the file.
- The calling process was attempting to change auditing flags. However, the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges.
- The calling process was attempting to change the security auditor's auditing flags, but the user does not have auditor authority.
- Attributes indicate that the security label is to be set, and one or more of the following conditions apply:
  - The calling process does not have RACF SPECIAL authorization and appropriate privileges.
  - A security label is already associated with the file.

**ERANGE**

The length of the extended attribute value to be set, AttxValLn, does not match the attribute size itself. For example, ERANGE can occur when a 4-byte attribute is being set to a 2-byte or 6-byte value. The following reason code can accompany the return code: JrInvAttrValLen.

**EROFS**

Pathname specifies a file that is on a read-only file system. Consult the reason code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the chattr service stores the reason code. The chattr service returns a reason code only if the return value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Changing extended attributes**

To add, remove, or modify extended attributes, use the extended attributes control block ATTX in place of the standard ATT control block. For mapping to ATTX, see [“BPXYATT — Map file attributes for chattr and fchattr” on page 1057](#). Rather than turning on flags within the control block like ATT, ATTX supports an attribute name field, AttxName, to specify which attribute to take action against. The ATTX mapping is variable length with the attribute value, AttxVal, field being a CHAR(\*) value. The AttxValLn field should be set to the length of the attribute value.

- The subcommand field, AttxSubcmd, controls whether this service call will set or remove the attribute. If GET, LIST, or any other value is specified, then the call will fail with JRInvAttxCmd.
- AttxFlags, AttxfReplace, and AttxfCreate are used to validate the request against the extended attribute. AttxfReplace checks if the attribute is set and can be replaced. If it is not set, this service call returns reason code JrAttxUnavail. AttxfCreate checks if the attribute is not set and able to be created (set). If it is already set, BPX1CHR/BPX4CHR returns reason code JrAttxExists.
- The AttxfLset flag controls whether the operation affects a symlink's attributes or the attributes of the symlink's target. When this flag is ON, the operation affects the symlink. When the flag is OFF, the operation affects the symlink's target.
- The SET/REMOVE function affects the attribute information that also exists in BPXYSTAT. The following table provides a correspondence between the information in BPXYATT and BPXYSTAT.

<i>Table 2. Correspondence between BPXYATT and BPXYSTAT</i>			
<b>Xattr name</b>	<b>Existing SET attribute in BPXYATT</b>	<b>Existing GET attribute in BPXYSTAT</b>	<b>Restrictions and comments</b>
trusted.apfauth	attapfauth	ST_APFAUTH	Set and remove functions that are limited according to "general flags" restrictions.
trusted.sharelib	attsharelib	ST_SHARELIB	Set and remove functions that are limited according to "general flags" restrictions.
trusted progctl	attprogctl	ST_PROGCTL	Set and remove functions that are limited according to "general flags" restrictions.
system.noshareas	attnoshareas	ST_NOSHAREAS	Set and remove functions that are limited according to "general flags" restrictions.
system.filefmt	attfilefmt	ST_FILEFMT	None.
system.filetag	attfiletag	ST_FILETAG	None.
system.seclabel	attseclabel	ST_SECLABEL	None.
system.useraudit	attuseraudit	ST_USERAUDIT	None.
system.auditoraudit	attauditoraudit	ST_AUDITORAUDIT	None.
system.auditid	Not an attribute. Equivalent to field ST_AUDITID in BPXYSTAT.	ST_AUDITID	Read-only.
system.dmodelacl	Not an attribute. Equivalent to field.	ST_DMODELACL	Read-only.
system.fmodelacl	Not an attribute. Equivalent to field.	ST_FMODELACL	Read-only.
system.accessacl	Not an attribute. Equivalent to field.	ST_ACCESSACL	Read-only.
system.createtime	Not an attribute. Equivalent to field.	ST_CREATETIME	Read-only.
user.*	No equivalent.	No equivalent.	Needs specific PFS support.

## Usage notes

<i>Table 3. Attribute fields that can be modified by chattr</i>		
Set flags	Attribute fields input	Description
<b>ATTMODECHG</b>	ATTMODE	Set the mode according to the value in ATTMODE. See <a href="#">“chmod (BPX1CHM, BPX4CHM) – Change the mode of a file or directory”</a> on page 101.
<b>ATTOWNERCHG</b>	ATTUID ATTGID	Set the owner user identifier (UID) and group identifier (GID) to the values specified in ATTUID and ATTGID. See <a href="#">“chown (BPX1CHO, BPX4CHO) – Change the owner or group of a file or directory”</a> on page 104.
<b>ATTSETGEN</b>	ATTGENVALUE ATTGENMASK	Only the bits corresponding to the bits set ON in the ATTGENMASK are set to the value (ON or OFF) in ATTGENVALUE. Other bits are unchanged.
<b>ATTTRUNC</b>	ATTSIZE	Change the file size to ATTSIZE bytes. See <a href="#">“ftruncate (BPX1FTR, BPX4FTR) – Change the size of a file”</a> on page 249.
<b>ATTATIMECHG</b>	ATTATIME	If ATTLTP64TIMES is not set, set the access time of the file to the value specified in ATTATIME. If ATTLTP64TIMES is set, set the access time of the file to the value specified in ATTATIME64, which is a doubleword field.
<b>ATTATIMETOD</b>	None	Set the access time of the file to the current time.
<b>ATTMTIMECHG</b>	ATTMTIME	If ATTLTP64TIMES is not set, set the modification time of the file to the value specified in ATTMTIME. If ATTLTP64TIMES is set, set the modification time of the file to the value specified in ATTMTIME64, which is a doubleword field.
<b>ATTMTIMETOD</b>	None	Set the modification time of the file to the current time.
<b>ATTMAAUDIT</b>	ATTAUDITORAUDIT	Set the security auditor's auditing flags to the value specified in ATTAUDITORAUDIT. See <a href="#">“chaudit (BPX1CHA, BPX4CHA) – Change audit flags for a file by path”</a> on page 95.
<b>ATTMUAUDIT</b>	ATTUSERAUDIT	Set the User's auditing flags to the value specified in ATTUSERAUDIT. See <a href="#">“chaudit (BPX1CHA, BPX4CHA) – Change audit flags for a file by path”</a> on page 95.

Table 3. Attribute fields that can be modified by chattr (continued)

Set flags	Attribute fields input	Description
<b>ATTCTIMECHG</b>	ATTCTIME	If ATTLP64TIMES is not set, set the change time of the file to the value specified in ATTCTIME. If ATTLP64TIMES is set, set the change time of the file to the value specified in ATTCTIME64, which is a doubleword field.
<b>ATTCTIMETOD</b>	None	Set the change time of the file to the current time.
<b>ATTREFTIMECHG</b>	ATTREFTIME	If ATTLP64TIMES is not set, set the reference time of the file to the value specified in ATTREFTIME. If ATTLP64TIMES is set, set the reference time of the file to the value specified in ATTREFTIME64, which is a doubleword field.
<b>ATTREFTIMETOD</b>	None	Set the reference time of the file to the current time.
<b>ATTFILEFMTCHG</b>	ATTFILEFMT	Set the File Format of the file to the value specified in ATTFILEFMT.
<b>ATTCHARSETIDCHG</b>	ATTFILETAG	Set the file tag. See BPXYSTAT ( <a href="#">“BPXYSTAT — Map the response structure for stat” on page 1162</a> ) for file tag mapping.
<b>ATTSECLABELCHG</b>	ATTSECLABEL	Set the initial security label for a file or directory.

1. Flags in the Attributes parameter are set to indicate which attributes are to be updated. To set an attribute, turn the corresponding Set Flag on, and set the corresponding Attributes field according to Table 3 on page 87. Multiple attributes can be changed at the same time.

Clear the **Set Flag** field before turning on any bits. It is an error if any of the reserved bits in the flag field are turned on.

2. Some of the attributes that are changed by the chattr service can also be changed by other services. See the related service (listed in Table 3 on page 87) for a detailed description.

### 3. Changing mode (ATTMODECHG = ON):

- The file mode field in Attributes is mapped by the BPXYMODE macro (see [“BPXYMODE — Map the mode constants” on page 1106](#)). For information about the values for file type, see [“BPXYFTYP — File type definitions” on page 1078](#).
- File descriptors that are open when the chattr service is called retain the access permission they had when the file was opened.
- The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges (see [“Authorization” on page 8](#)).
- Setting the set-group-ID-on-execution permission (in mode) means that when this file is run through the exec, attach\_exec, or spawn service, the effective GID of the caller is set to the file's owner GID. The caller will seem to be running under the GID of the file rather than that of the actual invoker.

The set-group-ID-on-execution permission is set to zero if the caller does not have appropriate privileges and the GID of the file's owner does not match the effective GID or one of the supplementary GIDs of the caller.

- Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

#### 4. Changing owner (ATTOWNERCHG = ON):

To change the owner UID of a file, the caller must have appropriate privileges.

To change the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:

- The effective UID of the caller matches the file's owner UID.
- The Owner\_UID value that is specified in the change request matches the file's owner UID.
- The Group\_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.

When the owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.

When the owner is changed, both UID and GID must be specified as they are to be set, or set to -1 if the value is to remain unchanged. If only one of these values is to be changed, the other can be set to its present value or to -1 to remain unchanged.

#### 5. Changing General Attribute bits (ATTSETGEN = ON):

- For General Attribute bits to be changed, the calling process must have write permission for the file.

#### 6. Changing the file size (ATTTRUNC = ON):

- The resizing of a file to ATTSIZE bytes changes the file size to ATTSIZE, beginning from the first byte of the file. If the file was originally larger than ATTSIZE bytes, the data from ATTSIZE to the original end of file is removed. If the file was originally shorter than ATTSIZE, bytes between the old and new lengths are read as zeros.

Full blocks are returned to the file system so that they can be used again.

The file offset is not changed.

- When a file size is changed successfully, it clears the set-user-ID, the set-group-ID, and the save-text (sticky bit) attributes of the file, unless the caller has appropriate privileges.
- The resizing of a file to ATTSIZE bytes, where ATTSIZE is greater than the soft file size limit for the process, fails with EFBIG, and the SIGXFSZ signal is generated for the process.
- A file's size cannot be changed if it is open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#) for details about the NFS share reservations.

#### 7. Changing times:

- All of the time fields in Attributes are in POSIX format.
- For the access time or the modification time to be set explicitly (ATTATIMECHG = ON or ATTMTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the access time or modification time to be set to the current time (ATTATIMETOD = ON or ATTMTIMETOD = ON), the effective ID must match the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.
- For the change time or the reference time to be set explicitly (ATTCTIMECHG = ON or ATTREFTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the change time or reference time to be set to the current time (ATTCTIMETOD = ON or ATTREFTIMETOD = ON), the calling process must have write permission for the file.

- For the any time field (**atime**, **mtime**, **ctime**, **reftime**), if both current time and specific time are requested (for example, ATTCTIMETOD = ON and ATTCTIMECHG = ON), the current time is set.
- When any attribute field is changed successfully, the file's change time is also updated.

**8. Changing auditor audit flags (ATTMAAUDIT = ON):**

- For auditor audit flags to be changed, the user must have auditor authority. Users with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

You establish auditor authority by issuing the TSO/E command ALTUSER Auditor.

**9. Changing user audit flags (ATTMUAUDIT = ON):**

- For the user audit flags to be changed, the user must have appropriate privileges (see [“Authorization” on page 8](#)) or be the owner of the file.

**10. Changing the file format (ATTFILEFMTCHG = ON):**

- The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
- The attribute that is specified in ATTFILEFMT is the same attribute that is set by the FILEDATA=TEXT parameter on a DD statement.

**11. Changing the file tag (ATTCHARSETIDCHG=ON):**

- A file tag can be set for regular, FIFO, and character special files. If the DeferTag bit is on in the file tag, the file must be empty.
- The tagging of /dev/null, /dev/random, /dev/urandom, and /dev/zero is ignored.

**12. Changing the security label (ATTSECLABELCHG=ON):**

- For the security label to be changed, the user must have RACF SPECIAL authorization and appropriate privileges (see [“Authorization” on page 8](#)), and no security label must currently exist on the file. Only an initial security label can be set. An existing security label cannot be changed. The function will successfully set the security label if the RACF SECLABEL class is active. If the SECLABEL class is not active, a return code of EMVSERR will be returned.

**Related services**

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“chaudit \(BPX1CHA, BPX4CHA\) — Change audit flags for a file by path ” on page 95](#)
- [“fchattr \(BPX1FCR, BPX4FCR\) — Change the attributes of a file or directory by descriptor” on page 190](#)
- [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file” on page 249](#)
- [“lchattr \(BPX1LCR, BPX4LCR\) — Change the attributes of a file or directory or symbolic link” on page 371](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)
- [“truncate \(BPX1TRU, BPX4TRU\) — Change the size of a file ” on page 953](#)
- [“utime \(BPX1UTI, BPX4UTI\) — Set file access and modification times ” on page 981](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

**Characteristics and restrictions**

1. The ATTEXTLINK flag in the **ATTGENVALUE** field of BPXYATT cannot be modified with BPX1CHR (BPX4CHR).
2. The security label (ATTSECLABELCHG) flag requires RACF SPECIAL authorization and appropriate privileges (see [“Authorization” on page 8](#)). It cannot be used to change an existing security label; it can only be used to set an initial security label on a file.



## Examples

See “BPX1CHR (chattr) example” on page 1228 and “BPX4CHR (chattr) example” on page 1321.

# chattrat (BPX1CRA, BPX4CRA) – Change the attributes of a file or directory relative to a directory file descriptor

---

## Function

The chattrat service modifies the attributes that are associated with a file or directory relative to a directory file descriptor. You can change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, file size, and file tag. You can also use it to set the initial security label for a file or directory.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CRA)</b>	31-bit
<b>AMODE (BPX4CRA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CRA,(File_descriptor,
              Pathname_length,
              Pathname
              Attribute_length,
              Attribute,
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CRA with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### File\_descriptor

Supplied parameter.

#### Type

File descriptor.

#### Length

Fullword.

The name of a fullword that contains the file descriptor of the file whose attributes that you want to change.

**Pathname\_length**

Supplied parameters.

**Type**

Character string.

**Length**

Fullword.

The name of a fullword that contains the length of the path name of the file whose attributes that you want to change.

**Pathname**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Variable.

The name of a field that contains the path name of the file. The length of this field is specified in Pathname\_length.

**Attributes\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the length of the area that contains the attributes you want to change.

**Attributes**

Supplied parameter.

**Type**

Structure.

**Length**

Specified by the Attributes\_length parameter.

The name of the area that contains the attributes that you want to change. The area is mapped by BPXYATT. For more information about the content of this area, see [“BPXYATT — Map file attributes for chatr and fchatr”](#) on page 1057.

**Flags**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The flags field is a bit mask that can be 0, or include the following constant, which is defined in <fcntl.h>:

**AT\_SYMLINK\_NOFOLLOW**

If *pathname* specifies a symbolic link, then update the symlink attribute of the link instead of fixing the file to which it refers.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword where the chattrat service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the chattrat service stores the return code. The chattrat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The chattrat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	<p>The calling process did not have appropriate permissions. Possible reasons include:</p> <ul style="list-style-type: none"> <li>• The calling process was attempting to set access time or modification time to current time, but the effective UID of the calling process does not match the owner of the file. The process does not have write permission for the file or the process does not have appropriate privileges.</li> <li>• The calling process was attempting to truncate the file, and it does not have write permission for the file.</li> </ul>
<b>EBADF</b>	<ul style="list-style-type: none"> <li>• <i>pathname</i> is relative but <i>dirfd</i> is not AT_FDCWD or a valid file descriptor.</li> <li>• <i>fd</i> is not a valid file descriptor.</li> </ul>
<b>EBUSY</b>	<p>The file is open by a remote NFS client with a share reservation that conflicts with the requested operation.</p>
<b>EFBIG</b>	<p>The calling process tried to change the size of a file, but the specified length is greater than the maximum file size limit for the process. See the reason code to determine the reason for the error. The following reason code can accompany the return code: JRWriteBeyondLimit</p>
<b>EINVAL</b>	<p>One of the following situations occurred:</p> <ul style="list-style-type: none"> <li>• Invalid value in <i>flags</i>.</li> <li>• <i>pathname</i> is NULL, <i>dirfd</i> is not AT_FDCWD, and flags contains AT_SYMLINK_NOFOLLOW.</li> <li>• The length of the Attributes parameter is too small or the Attributes structure that has the requested changes is not valid. See the reason code to determine the reason for the error.</li> </ul> <p>The following reason codes can accompany the return code: JrInvalidAtt, JrNegativeValueInvalid, JrTrNotRegFile, JrTrNegOffset, JrFileNotEmpty, and JrInvalidFileTag.</p>

Return_code	Explanation
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMVSERR</b>	An environmental error was detected. The following reason code can accompany the return code: JrSeclabelClassInactive.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters. File name truncation is not supported.
<b>ENOENT</b>	A file with the name <i>Pathname</i> was not found, or a path name was not specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOSYS</b>	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.
<b>ENOTDIR</b>	Some component of Pathname is not a directory.
<b>EPERM</b>	<p>The operation is not permitted for one of the following reasons.</p> <ul style="list-style-type: none"> <li>• The calling process tried to change the mode or the file format, but the effective UID of the calling process does not match the owner of the file. The calling process also does not have appropriate privileges. Refer to <a href="#">“Authorization”</a> on page 8 for information on appropriate privileges.</li> <li>• The calling process tried to change the owner, but it does not have appropriate privileges.</li> <li>• The calling process tried to change the general attribute bits, but it does not have write permission for the file.</li> <li>• The calling process tried to set a time value (not current time). However, the effective user ID does not match the owner of the file and it does not have appropriate privileges.</li> <li>• The calling process tried to set the change time or reference time to current time, but it does not have write permission for the file.</li> <li>• The calling process tried to change auditing flags, but the effective UID of the calling process does not match the owner of the file. Also, the calling process does not have appropriate privileges.</li> <li>• The calling process tried to change the security auditor's auditing flags, but the user does not have auditor authority.</li> <li>• Attributes indicate that the security label is to be set. One or more of the following conditions must also apply: <ul style="list-style-type: none"> <li>– The calling process does not have RACF® SPECIAL authorization and appropriate privileges.</li> <li>– A security label is already associated with the file.</li> </ul> </li> </ul>
<b>EROFS</b>	<i>Pathname</i> specifies a file that is on a read-only file system. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRReadOnlyFS.
<b>Reason_code</b>	Returned parameter.
<b>Type</b>	Integer
<b>Length</b>	Fullword

The name of a fullword in which the chatrat service stores the reason code. The chatrat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

None.

## Related services

- [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory ” on page 82](#)
- [“fchattr \(BPX1FCR, BPX4FCR\) — Change the attributes of a file or directory by descriptor” on page 190](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1CRA \(chatrat\) example” on page 1230](#) and [“BPX4CRA \(chatrat\) example” on page 1323](#).

# chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path

---

## Function

The chaudit service changes the types of access to a file to be audited for the security product. The chaudit service identifies the file by its path name.

For the corresponding service using a file descriptor, see [“fchaudit \(BPX1FCA, BPX4FCA\) — Change audit flags for a file by descriptor” on page 199](#).

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CHA):</b>	31-bit
<b>AMODE (BPX4CHA):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CHA, (Pathname_length,
               Pathname,
               Audit_flags,
```

```

Option_code,
Return_value,
Return_code,
Reason_code)

```

AMODE 64 callers use BPX4CHA with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the path name of the file.

### Pathname

Supplied parameter

#### Type:

Character string

#### Character set:

No restriction

#### Length:

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file for which auditing is to be changed.

Path names can begin with or without a slash:

- A path name that begins with a slash is an absolute pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative pathname. The search for the file starts at the working directory.

### Audit\_flags

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that indicates the access to be audited. This field is mapped by the BPXYAUDT macro ([“BPXYAUDT — Map flag values for chaudit and fchaudit” on page 1060](#)). Valid values for this field include any combination of the following:

#### AUDTREADFAIL

Audit requests that are failing.

#### AUDTREADSUCCESS

Audit successful read requests.

#### AUDTWRITEFAIL

Audit write requests that are failing.

#### AUDTWritesUCCESS

Audit successful write requests.

#### AUDTEXECFAIL

Audit execute or search requests that are failing.

**AUDTEXECSUCCESS**

Audit successful execute or search requests.

**Option\_code**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that indicates whether you are changing the auditing for the user or for the security auditor. When this field has the value:

- 0, the user's auditing is being changed.
- 1, the security auditor's auditing is being changed. A superuser who is not the auditor cannot change the auditor's authority.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chaudit service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chaudit service stores the return code. The chaudit service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The chaudit service returns one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have search permission for some component of the Pathname prefix.
<b>EINVAL</b>	The Option_code parameter is incorrect. The following reason code can accompany the return code: JRBadAuditOption.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or some component of the path name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	A component of the Pathname prefix is not a directory.

Return_code	Explanation
<b>EPERM</b>	The effective UID of the calling process does not match the file's owner UID; the calling process does not have appropriate privileges (see <a href="#">“Authorization” on page 8</a> ); or if Option_code indicated that the auditor audit flags were to be changed, the user does not have auditor authority.
<b>EROFS</b>	The file exists on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chaudit service stores the reason code. The chaudit service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. If Option\_code indicates that the auditor audit flags are to be changed, the user must have auditor authority for the request to be successful. The user with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

You can get auditor authority by entering the TSO/E command ALTUSER Auditor.

2. If Option\_code indicates that the user audit flags are to be changed, the user must have appropriate privileges (see [“Authorization” on page 8](#)) or be the owner of the file.

**Related services**

- [“fchaudit \(BPX1FCA, BPX4FCA\) — Change audit flags for a file by descriptor” on page 199](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name” on page 893](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1CHA \(chaudit\) example” on page 1225](#).

## chdir (BPX1CHD, BPX4CHD) — Change the working directory

---

**Function**

The chdir service changes your working directory from the current one to a new one. The working directory is the starting point for path searches of path names that do not begin with a slash.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task



Operation	Environment
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CHD)</b>	31-bit
<b>AMODE (BPX4CHD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CHD, (Pathname_length,
               Pathname,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CHD with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the path name of the directory that is to become your new working directory.

### Pathname

Supplied parameter

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the new directory. This field has the length that is specified in Pathname\_length.

Path names can begin with or without a slash:

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory.

### Return\_value

Returned parameter

**Type**

Integer

**Length:**

Fullword

The name of a fullword in which the chdir service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length:**

Fullword

The name of a fullword in which the chdir service stores the return code. The chdir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The chdir service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The calling process does not have permission to search one of the components of Pathname.
<b>EINVAL</b>	The Pathname parameter is not valid; it contains nulls.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of Pathname is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No directory named Pathname was found, or no Pathname was specified. The following reason codes can accompany the return code: JRChdNoEnt and JRQuiesting.
<b>ENOTDIR</b>	Some component of Pathname is not a directory. The following reason code can accompany the return code: JRChdNotDir.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the chdir service stores the reason code. The chdir service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory” on page 124](#)
- [“chroot \(BPX1CRT, BPX4CRT\) — Change the root directory” on page 111](#)
- [“fchdir \(BPX1FCD, BPX4FCD\) — Change the working directory ” on page 202](#)
- [“getcwd \(BPX1GCW, BPX4GCW\) — Get the path name of the working directory” on page 260](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)

- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory” on page 523](#)
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory ” on page 656](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory ” on page 703](#)
- [“realpath \(BPX1RPH, BPX4RPH\) — Resolve a path name” on page 677](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1CHD \(chdir\) example” on page 1226](#).

# chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory

---

## Function

The chmod service modifies the permission bits that are used to control the owner access, group access, and general access to a file. You can use it to set flags that modify the user ID (UID) and group ID (GID) of the file when it is executed. You can also use it to set the sticky bit to indicate from where the file should be fetched. You identify the file by its path name.

For the corresponding service using a file descriptor, see [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor” on page 204](#).

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CHM)</b>	31-bit
<b>AMODE (BPX4CHM)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CHM, (Pathname_length,
               Pathname,
               Mode,
               Return_value,
```

Return_code, Reason_code)
------------------------------

AMODE 64 callers use BPX4CHM with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the path name of the file whose mode you want to change.

### Pathname

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. This field has the length that is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory.

### Mode

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword that describes the access. This field, which is mapped by BPXYMODE, specifies the file type and permissions for the caller, for the callers group, and for any others. For more information, see [“BPXYMODE — Map the mode constants ” on page 1106.](#)

### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chmod service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chmod service stores the return code. The chmod service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The chmod service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search some component of Pathname.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters. Filename truncation is not supported.
<b>ENOENT</b>	No file that was named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of Pathname is not a directory.
<b>EPERM</b>	The effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges. (See <a href="#">“Authorization”</a> on page 8.)
<b>EROFS</b>	Pathname specifies a file that is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chmod service stores the reason code. The chmod service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. File descriptors that are open when the chmod service is called retain the access permission they had when the file was opened.
2. For mode bits to be changed, the effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges (see [“Authorization”](#) on page 8).
3. A user with READ authority to SUPERUSER.FILES.CHANGEPERMS in the UNIXPRIV class can use the chmod service to change the permission bits of any file.
4. When the mode is changed successfully, the file's change time is also updated.
5. Setting the set-group-ID-on-execution permission means that when this file is run, through the exec, spawn, or attach\_exec service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file rather than that of the actual invoker.

The set-group-ID-on-execution permission is set to zero if both of the following are true:

- The caller does not have appropriate privileges.
  - The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.
6. Setting the set-user-ID-on-execution permission means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

**Related services**

- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor” on page 204](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1CHM \(chmod\) example” on page 1226](#).

## **chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory**

---

**Function**

The chown service changes a file's owner, group, or both owner and group. The owner is identified by a user ID (UID) and a group ID (GID).

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CHO)</b>	31-bit
<b>AMODE (BPX4CHO)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1CHO, (Pathname_length,
               Pathname,
               Owner_UID,
               Group_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CHO with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the path name of the file whose owner or group is to be changed.

### Pathname

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. This field has the length that is specified in Pathname\_length.

Path names can begin with or without a slash:

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory.

### Owner\_UID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that contains the new owner UID that is assigned to the file. If there is no change, this field contains the present value or -1. This parameter must be specified.

### Group\_ID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that contains the new owner GID that is assigned to the file. If there is no change, this field contains the present value or -1. This parameter must be specified.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chown service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chown service stores the return code. The chown service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The chown service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search some component of the Pathname prefix.
<b>EINVAL</b>	The Owner_UID or Group_ID parameter is incorrect.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters.
<b>ENOENT</b>	No file named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of the Pathname prefix is not a directory.
<b>EPERM</b>	The calling process does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).
<b>EROFS</b>	Pathname is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chown service stores the reason code. The chown service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



## Usage notes

1. The chown service changes the owner UID and owner GID of a file. Only a caller with appropriate privileges can change the owner UID of a file. Refer to [“Authorization” on page 8](#) for information on appropriate privileges.
2. The owner GID of a file can be changed by a caller if the caller has appropriate privileges, or if a caller meets all of these conditions:
  - The effective UID of the caller matches the file's owner UID.
  - The Owner\_UID value that is specified in the change request matches the file's owner UID.
  - The Group\_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner\_UID and Group\_ID must be specified. To change only one of these values, set the one that is to remain unchanged to its present value or to -1.

## Related services

- [“fchown \(BPX1FCO, BPX4FCO\) — Change the owner and group of a file or directory by descriptor” on page 210](#)
- [“lchown \(BPX1LCO, BPX4LCO\) — Change the owner or group of a file, directory, or symbolic link” on page 380](#)
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#)
- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1CHO \(chown\) example” on page 1226](#).

# chpriority (BPX1CHP, BPX4CHP) — Change the scheduling priority of a process

---

## Function

The chpriority callable service changes the scheduling priority of a process, process group, or user.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CHP):</b>	31-bit

Operation	Environment
<b>AMODE (BPX41CHP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CHP,(Which,
               Who,
               PriorityType,
               Priority,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CHP with the same parameters.

## Parameters

### Which

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates how the Who parameter is to be interpreted. This parameter can have one of the following values:

#### **PRIO\_PROCESS**

Indicates that the Who parameter is to be interpreted as a process ID.

#### **PRIO\_PGRP**

Indicates that the Who parameter is to be interpreted as a process group ID.

#### **PRIO\_USER**

Indicates that the Who parameter is to be interpreted as a user ID.

The PRIO\_ constants are defined in the BPXYCONS macro.

### Who

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates the exact process ID, process group ID, or user ID whose priority is to be changed. The Which parameter indicates how this parameter is to be interpreted. If this parameter is interpreted as a process group ID or user ID, all processes with the specified process group ID or user ID are to have their priority changed. A value of zero for this parameter specifies the current process, process group, or user ID.

### PriorityType

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains a value that indicates how the Priority parameter is to be interpreted. This parameter can have one of the following values:

**CPRIO\_ABSOLUTE**

Indicates that the Priority parameter is to be interpreted as an absolute value. This causes the priority value of the target processes to be set to the value specified by the Priority parameter.

**CPRIO\_RELATIVE**

Indicates that the Priority parameter is to be interpreted as a relative value. This causes the priority value of the target processes to be incremented or decremented by the value that is specified by the Priority parameter.

The CPRIO\_ constants are defined in the BPXYCONS macro.

**Priority**

Supplied parameter

**Type:**

Signed Integer

**Length:**

Fullword

The name of a fullword that contains a value that indicates the priority value that the specific process or group of processes is to be set to or changed by.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chpriority service returns -1 if it is not successful. If it is successful, the chpriority service returns a value of zero.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chpriority service stores the return code. The chpriority service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The chpriority service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The priority is being changed to a lower value, and the current process does not have the appropriate privilege (see <a href="#">“Authorization” on page 8</a> ) to do so.
<b>EPERM</b>	A process was located, but the saved set-user-ID of the calling process does not match the saved set-user-ID of the process whose priority is being changed.

Return_code	Explanation
<b>EINVAL</b>	The value of the Which parameter was not recognized; the value of the Who parameter is not a valid process ID, process group ID or user ID; or the value of the PriorityType parameter is not supported.
<b>ESRCH</b>	No process could be located using the Which and Who parameter values specified.
<b>EMVSSAF2ERR</b>	A Security product internal error has occurred. Consult the Reason_code parameter for the exact reason for the error.
<b>ENOSYS</b>	The system does not support this function. Your installation has chosen not to enable this function.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chpriority service stores the reason code. The chpriority service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the supplied Who and Which values specify more than one process, each of the specified processes has its priority values set to the supplied value. If at least one of the specified processes has its priority value successfully changed, the chpriority service returns successfully.
2. The priority value of a process is an integer that can be in the range of -20 to 19. If the priority value that is supplied causes the priority value of a process to be outside this range, the priority of the process is set to the corresponding limit value. The default priority value for all processes is 0.
3. An increase in the priority value of a process results in a lower CPU priority for the process. A decrease in the priority value of a process results in a higher CPU priority for the process.
4. If the supplied priority value would result in a lower priority value for the specified processes, the caller must have appropriate privileges. Refer to [“Authorization” on page 8](#) for information about appropriate privileges. In addition to being able to lower the priority value, a caller with appropriate privileges can change the priority of any other process, regardless of the saved set-user-ID value of the process.
5. The setting of the priority value of a process has a corresponding effect on its nice value, as they both represent the relative CPU priority of the process. For example, if you use the chpriority service to change the priority value of a process to its maximum value (19), the nice value of the process is changed to its maximum value (2\*NICE\_ZERO)-1. This is reflected on the nice, getpriority, chpriority and setpriority services. The NICE\_ZERO constant is defined in BPXYCONS.
6. If the ENOSYS return code is received, your installation does not support this service. Contact your system administrator if you require activation of this service.
7. If the supplied Who and Which values specify a process in a multiple—process address space, each of the processes in the address space will have their priority values set to the supplied value.
8. For information about the necessary system setup for this service, [Enabling nice\(\)](#), [setpriority\(\)](#), and [chpriority\(\)](#) support in *z/OS UNIX System Services Planning*.

**Related services**

- [“nice \(BPX1NIC, BPX4NIC\) — Change the nice value of a process” on page 497](#)

- “[getpriority \(BPX1GPY, BPX4GPY\)](#) — Get the scheduling priority of a process ” on page 300
- “[setpriority \(BPX1SPY, BPX4SPY\)](#) — Set the scheduling priority of a process” on page 779

## Characteristics and restrictions

None.

## Examples

See “[BPX1CHP \(chpriority\) example](#)” on page 1227.

# chroot (BPX1CRT, BPX4CRT) – Change the root directory

---

## Function

The chroot service changes the root directory from the current one to a new one. The root directory is the starting point for path searches of path names beginning with a slash. The working directory of the process is unaffected by chroot().

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CRT)</b>	31-bit
<b>AMODE (BPX4CRT)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CRT, (Pathname_length,
               Pathname,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CRT with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the path name of the directory that is to become your root directory.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the new directory. This field has the length that is specified in Pathname\_length.

Path names can begin with or without a slash:

- A path name that begins with a slash is an *absolute* path name. The slash refers to the current root directory, and the search for the file starts at the current root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chroot service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chroot service stores the return code. The chroot service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The chroot service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search one of the components of Pathname.
<b>EINVAL</b>	The Pathname parameter is not valid; it contains nulls.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of Pathname is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No directory named Pathname was found, or no Pathname was specified. The following reason codes can accompany the return code: JRChdNoEnt and JRQuiescing.

Return_code	Explanation
<b>ENOTDIR</b>	Some component of Pathname is not a directory. The following reason code can accompany the return code: JRChdNotDir.
<b>EPERM</b>	The calling process is not a superuser. The following reason code can accompany the return code: JRUserNotPrivileged.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the chroot service stores the reason code. The chroot service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. Upon completion of the chroot, the specified directory is now the logical root of the file system for the process. All searches for path name beginning with slash (/) start from this directory, and all attempts to use dot dot (..) over the root remain in the new root.
2. A new child process inherits a parent's changed root directory.
3. If the current working directory is above the new root, chroot(.) can be used to reset the root directory to equal the current working directory. However, when the current working directory is above the root directory, getcwd() fails with ENOENT return code.

**Related services**

- [“chdir \(BPX1CHD, BPX4CHD\) — Change the working directory ” on page 98](#)
- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory” on page 124](#)
- [“getcwd \(BPX1GCW, BPX4GCW\) — Get the path name of the working directory” on page 260](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory” on page 523](#)
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory ” on page 656](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory ” on page 703](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)

**Characteristics and restrictions**

To change the root directory, the caller must have appropriate privileges (see [“Authorization” on page 8](#)).

**Examples**

See [“BPX1CRT \(chroot\) example” on page 1230](#).

## clone (BPX1CLN, BPX4CLN) – Create a child process

---

### Function

The clone service creates a new process, which is called a *child process*. This service is similar to the fork callable service but has more options such as allowing the new child process to run in a newly created namespace.

For the macro, see [“BPXYCLNP— Map clone syscall parameters” on page 1062](#).

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, PSW key 8, TCB key 8
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CLN)</b>	31-bit
<b>AMODE (BPX4CLN)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1CLN, (CLNP_length,
               CLNP,
               Process_ID,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CLN with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

### Parameters

#### Clnp\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that contains the length of the Clnp control block that is being passed in the next parameter. To determine the value of Clnp\_length, use the BPXYCLNP macro. (See [“BPXYCLNP— Map clone syscall parameters” on page 1062](#).)

#### Clnp

Supplied parameter.

#### Type

Structure.



**Length**

Specified by the `Clnp_length` parameter.

The name of a CLNP structure that is to be used to control the extent of the sharing between the calling process and the child process. See section on the CLNP control block in the usage notes for details on setting the fields of the CLNP. The BPXYCLNP macro maps the CLNP. (See [“BPXYCLNP—Map clone syscall parameters”](#) on page 1062.)

The following table shows the flags that can be specified.

Flag	Description
<b>0</b>	Creates a process, the same as fork.
<b>CLONE_NEWIPC</b>	Creates the process in a new IPC namespace.
<b>CLONE_NEWPID</b>	Creates the process in a new PID namespace. CLONE_NEWPID cannot be used with CLONE_PARENT.
<b>CLONE_PARENT</b>	Creates the process such that the parent of the new process is the same as the calling process. When the child is terminated, the parent of the calling process is signaled. CLONE_PARENT cannot be used with CLONE_NEWPID.

**Restriction:** The clone syscall accepts only a signal of SIGCHLD. Any other value will result in a EINVAL errno.

**Process\_ID**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the clone service places the process ID of the newly created child process, 0, or -1.

Upon successful completion, clone returns the process ID of the newly created child to the calling process.

Because the child is a duplicate, it contains the same service request to the clone service as the calling process. Execution of the child begins with this clone service returning a process ID value of zero. The child then proceeds with normal execution.

If `Process_ID` is returned as -1, no child process was created for the reason shown by `Return_code`.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the clone service stores the return code. The clone service returns `Return_code` only if `Process_ID` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* A list of possible return codes follows.

Return_code	Explanation
<b>EAGAIN</b>	<p>The resources required to allow another process be created are not available now, or you have already reached the maximum number of processes you can run.</p> <p>The following reason codes can accompany the return code: JR ForkExitRcChildNoStorage, JR ForkExitRcParentBadEnv, JR ForkExitRcParentNoRoom, JR ForkNoAccess, JR ForkNoResource, JR ForkVsmListTooLarge, JR KernelReady, JR MaxChild, JR MaxProc, JR MaxUIDs, JR NoSecurityProduct, JR NotKey8, and JR WlmWonErr.</p>
<b>EINVAL</b>	<p>One of the input parameters was not valid.</p> <ul style="list-style-type: none"> <li>• The identifier, version, and length values, signal, or clone_flags provided in the CLNP are incorrect.</li> <li>• A CLONE_PARENT request was made from the init process or a namespace init process.</li> </ul> <p>The following reason code can accompany the return code: JR JsRacXtr, JR CLNPNotValid, JR UnsupportedFlag, JR UnsupportedSignal, JR MutuallyExclFlag, Jr CalledFromInitProc.</p>
<b>ENOMEM</b>	<p>The process requires more space than is available.</p> <p>The following reason codes can accompany the return code: Jr NSInitProcTerm, Jr NamespaceNotFound.</p>
<b>ENOSPC</b>	<p>A system limit was reached.</p> <ul style="list-style-type: none"> <li>• The limit on the number of namespaces would be exceeded.</li> <li>• Creating a PID namespace as requested by CLONE_NEWPID would cause the nesting depth limit of PID namespaces to be exceeded.</li> </ul> <p>The following reason codes can accompany the return code: JR MaxNamespace, Jr MaxNamespaceNestIn.</p>
<b>EPERM</b>	<p>The calling process does not have appropriate privileges.</p> <ul style="list-style-type: none"> <li>• The user is not a superuser and is not permitted to the CONTAINERS resource in the UNIXPRIV class.</li> </ul> <p>The following reason code can accompany the return code: Jr NotAuthNameSp.</p>
<b>EMVSSAF2ERR</b>	<p>An error occurred in the security product.</p> <p>The following reason code can accompany the return code: Jr SAFInternal.</p>
<b>Reason_code</b>	
Returned parameter.	
<b>Type</b>	
Integer.	
<b>Length</b>	
Fullword.	
<p>The name of a fullword in which the clone service stores the reason code. The clone service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_value. For a list of reason codes, see <a href="#">Reason codes</a> in <i>z/OS UNIX System Services Messages and Codes</i>.</p>	

## Usage notes for clone

1. The input CLNP block must be set up correctly or the service will return with EINVAL. The identifier, version, and length values must all be set. The signal must be SIGCHLD.
2. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes the clone service, the child process that is created inherits the privilege of the set-user-ID or set-group-ID program.
3. The new process (the *child process*) is a duplicate of the process that calls the clone service (the *calling process*), with the following exceptions:
  - The child process has a unique process ID (PID) in its namespace and each of any ancestor namespaces that does not match any active process group ID.
  - The child is created in the same namespaces, unless one or more of the **CLONE\_NEWxxx** fields in the CLNP are set or a prior UNSHARE CLONE\_NEWPID or SETNS CLONE\_NEWPID was issued by the calling process.
  - The child has a different parent process ID (namely, the process ID of the process that called the clone service) unless CLONE\_PARENT was specified. If the new process is created in a PID namespace other than the PID namespace of the caller, the child appears to have no parent process (PPID=0) from its view within the namespace.
  - The child has its own copy of the calling process's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the calling process.
  - If the file has its FCTLCLOFORK flag set on, it is not inherited by the child process. This flag is set with the fcntl service. For more information, see [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#).
  - The child has its own copy of the calling process's open directory streams. Each open directory stream in the child can share directory stream positioning with the corresponding directory stream of the calling process.
  - The process and system utilization times for the child are set to zero.
  - Any file locks that were previously set by the calling process are not inherited by the child.
  - The child process has no interval timers set (similar to the results of a call to the alarm service with Wait\_time specified as zero).
  - The child has no pending signals.

In other respects, for z/OS UNIX the child is identical to the calling process.

4. When the clone requests the process to be added in one or more new namespaces (one of the CLONE\_NEWxxx flags was specified) the caller must be authorized by being a superuser or having at least READ access to the CONTAINERS resource in the UNIXPRIV class.
5. The child process inherits all key 8 shared memory segments that are attached to the calling process. The internal values of the number of processes that are attached to each shared memory segment (**shm\_nattch**) are incremented.

Because BPX1CLN only supports the propagation of key 8 storage, the clone service does not propagate to the child any shared memory segments that reside in a storage key other than key 8.

6. If the calling address space uses the macro IARV SERV to capture storage, the pages are not copied to the child address space.
7. The semaphore adjustment values (**semadj**) are cleared in the child process.
8. PSW Key 2 mmap storage areas are not propagated to the child. Above the bar key 2 and key 8 mmap storage areas are propagated to the child.
9. For AMODE 64 callers, high-memory storage is copied to the child process in the following cases:
  - All storage that is obtained by an IARV64 request that was made by the cloning thread is copied to the child process.

- All storage that is obtained by an IARV64 request with a user token that contains zeros in bits 0-31 and the calling process's PID in bits 32-64 is copied to the calling process. In the child process, the user token is changed to the value of the child process's PID in bits 32-64.
- All storage that is obtained by an IARV64 request with a user token that contains zeros in bits 0-31 and a nonzero value that matches ThliParentTkn in bits 32-64 (when ThliChildTkn is nonzero) is copied to the child process. In the child process, the user token is changed to the value of ThliChildTkn from the calling process. This value is also used to initialize ThliParentTkn on the child process.
- All authorized storage that is obtained by an IARV64 request with a user token that contains zeros in bits 32-64 and the calling process's PID in bits 0-31 is copied to the calling process. In the calling process, the user token is changed to the value of the child process's PID in bits 0-31.
- All authorized storage that is obtained by an IARV64 request with a user token that contains zeros in bits 32-64 and the value of PSALAA in bits 0-31 is copied to the child process. In the child process, the user token is changed to the value of the child process's LAA in bits 0-31.

10. The child process inherits the MEMLIMIT of the calling process.

The child address space inherits the following address space attributes of the calling process address space: region size and time limit.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ” on page 574](#)
- [“setrlimit \(BPX1SRL, BPX4SRL\) — Set resource limits ” on page 789](#)
- [“setns \(BPX1SNS, BPX4SNS\) — Reassociate thread with a namespace” on page 771](#)
- [“times \(BPX1TIM, BPX4TIM\) — Get process and child process times ” on page 950](#)
- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)
- [“unshare \(BPX1UNS, BPX4UNS\) — Run program in a new namespace” on page 977](#)

## Characteristics and restrictions

Following is a list of characteristics or restrictions for the clone service:

- The clone service can be requested from either an MVS or kernel address space.
- The clone service is supported from programs that are running in PSW key 8 only. An additional requirement is that the storage protection key value in the **TCBPKF** field of the task control block (TCB) must be 8. The clone service from authorized or problem-state programs with a PSW key other than 8 or a **TCBPKF** value other than 8 is rejected with an error code.
- A *namespace init process* is the first process to run within a PID namespace and is always assigned PID 1 within the namespace. A namespace init process is created as a result of a clone service request with CLONE\_NEWPID specified or the first child process created after an unshare service with CLONE\_NEWPID specified. When a namespace init process terminates, all processes in the PID namespace are signaled to terminate and no new processes can be created in the namespace.

Once the first child process is created after an unshare for a PID namespace, subsequent child processes created by the calling process are created in the new namespace. Similarly, a prior setns service for a PID namespace will cause subsequent children of the calling process to be created in a specific namespace. If child processes are to be created in a specific PID namespace whose init process has terminated, the clone service will fail with ENOMEM.

- The CLONE\_PARENT flag cannot be specified when invoked from a namespace init process. Doing so will result in a EINVAL error.
- Only the following storage subpools are copied by clone: 0-127, 129-132, and 251-252.
- Except for subpool 252, which is all key-0 storage, only the caller's key-8 storage is copied to the child. For subpools that support multiple keys (that is, subpool 129 to subpool 132) only storage that is obtained with a key of 8 is copied.
- When the clone service is called from a single-process address space, all storage that was obtained by all the tasks in the calling job step in the given subpools are copied to the child address space.

When the clone service is called from a multiple-process address space, only storage that is obtained by the tasks in the calling process in the previously identified subpools is copied to the child address space.

- The child process always runs in problem program state key of 8, even when it is cloned by an APF-authorized MVS process.
- One task (thread) and one request block (RB) are present in the child address space after the clone service request. If the calling process was single-task with multiple RBs, only a single RB is created in the child address space after the clone service request. If multiple tasks exist in the calling process, only the task issuing the clone service request is replicated. Serialization does not occur among the different tasks.
- The TCB address and the addresses of other MVS control blocks are likely to be different in the child.
- The clone service does not copy any system subpools or MVS control blocks from the calling process to the child, except as noted.

For example, the task I/O table (TIOT) is not copied. MVS data sets that were allocated in the calling process are not allocated to the child except for the propagated TASKLIB, STEPLIB, or JOBLIB DD data sets. Because user data in user subpools is copied, some of those control blocks might point to system control blocks that are no longer present in the child.

As another example, a user's data control block (DCB) that was opened in the calling process still appears as an opened DCB in the child. However, the corresponding system control blocks pointed to by the DCB are not present in the child.

Only services that are documented as supported can be used across the clone service.

- There is a limit on the total number of living or zombied children the calling process can have at a time. This limit is set with the MAXPROCUSER parameter in a BPXPRMxx parmlib member. You can retrieve this count with the sysconf service (BPX1SYN, BPX4SYN).
- There is a limit on the maximum number of namespaces (all types combined). It is set to one half of the maximum process limit. It is a static limit that is not affected by any changes that are made to MAXPROCSYS.
- There is a limit on the maximum number of namespaces in the system (all types combined). It is set to one half of the maximum process limit. It is a static limit that is not affected by any changes made to MAXPROCSYS.
- There is a limit on the number of processes allowed in a PID namespace (including processes in descendant namespaces as they are visible), set to one half the MAXPROCSYS when the namespace was created.
- PID namespaces can be nested, thus forming a hierarchical tree. There is a limit to the nesting depth of PID namespaces set to 4 namespaces levels underneath the root namespace.

Although the child process resembles the calling process in many ways, it has specific differences from the calling process. In addition to the differences described in POSIX.1 (under fork), the following are some examples of elements in the calling process that are not propagated to the child process:

#### Linkage stack

The caller can have a linkage stack, but the child does not inherit it. If the caller intends to do an exec service request in the child, the loss of the linkage stack is not a problem. It is a problem only if the child process executes a PR (Program Return) instruction that requires the linkage stack.

### **Access list (that is, PASN-AL, DU-AL)**

The calling process's access lists are not propagated to the child.

### **Access registers**

Access registers are not propagated to the child because the child process does not inherit the calling process's access list, which would be needed to use the access registers.

### **Virtual pages**

Virtual pages that were page-fixed in the calling process are not page-fixed in the child.

### **Dynamic resource managers (RESMGRs)**

Dynamic resource managers that were established for the calling process are not propagated to the child.

### **MVS files**

Any MVS files that were opened for the calling process are not opened for the child process, except for the TASKLIB, STEPLIB, or JOBLIB DD data sets that were propagated from the calling process. Only z/OS UNIX files are opened in the child process.

### **Mutexes and condition variables**

Because ownership of mutexes and condition variables is on a single-thread basis, these attributes cannot be propagated on clone. Where a mutex or condition variables exists, the thread that is created in the child has access to the shared memory and can use the mutex or condition variable. However, when it begins running, it will not own any mutexes or consume any condition variables.

### **Examples**

For examples that use the clone callable service, see [“BPX1CLN \(clone\) example” on page 1228](#) and [“BPX4CLN \(clone\) example” on page 1322](#).

## **MVS-related information**

1. Following is a list of services in the child that relate to the services done in the process that is being cloned.

### **GETMAIN, FREEMAN, or STORAGE**

If the calling process has issued a GETMAIN macro for a storage block, the child process can issue a FREEMAIN macro for the same storage block.

### **LOAD or DELETE**

If a problem state-calling process issues a LOAD macro for a module, the child process can issue a DELETE macro to remove the module from storage. If the child process issues a LOAD macro for the same module that was loaded in the calling process, the copied version of the module is used and the use count is incremented. If a supervisor state-calling process issues a LOAD macro for a module, the child process cannot issue a DELETE macro for the module. It also cannot use a LOAD macro to load a new copy of the module. However, a LOAD macro for global storage is not reflected in the child; the child cannot issue a DELETE macro to remove a module that was loaded to a common storage by the calling process.

### **CSVQUERY**

The EPTOKEN (entry point token) returned as OUTEPTKN on a CSVQUERY macro in the calling process can be used by the child as the INEPTKN parameter on a CSVQUERY macro to refer to the same module.

### **ESTAE**

The child process can issue an ESTAE macro with a 0 parameter to delete an ESTAE routine that was established by the calling process.

### **ESPIE**

The child process can delete an ESPIE routine that was established by the calling process. No other MVS services are carried across clone. They can be freely used in either the calling process or the child process. However, the result of these services (if performed in the calling process) cannot be available to the child process.

2. The system propagates the contents directory-related information (including extent lists) for the job pack queue for the job step task that is related to the task issuing the clone call . It also propagates the information on all modules (whether private or in the LPA) that have been loaded by the task issuing the clone call.
3. The system propagates the current task's SPIE or ESPIE and STAE or ESTAE status to the child process.
  - STAE or ESTAE control blocks that represent the current RB are propagated to the child process. Control blocks that are associated with older RBs are not propagated, nor are STAI or ESTAI control blocks.
  - SPIE or ESPIE control blocks that represent the current RB are propagated to the child process. SPIE or ESPIE control blocks that are associated with older RBs are not propagated.
4. Security information from the calling process's address space is propagated to the child's address space. As a result, the child has a security environment equivalent to that of the calling process.
5. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the current task are propagated to the child's address space. This causes the child address space to have the same MVS program search order as the calling process task.
6. The accounting information of the calling process's address space is propagated to the child's address space. (See [Managing accounting work in z/OS UNIX System Services Planning](#).)
 

If the ThliForkAcctg bit is set on in “BPXYTHLI — Thread-level information” on page 1165, the clone service creates the child with the accounting data from the RACF WORKATTR of the user ID that is associated with the last setuid call. If no setuid call has been performed, the accounting information from the calling process is used. No error is returned to the caller.
7. The job name of the calling process is propagated to the child and appended with a numeric value in the range of 1-9 if the job name is 7 characters or fewer. If the job name is 8 characters, the job name is propagated as is. When a job name is appended with a numeric value, the count wraps back to 1 when it exceeds 9.
8. If the calling process task is in a workload management (WLM) enclave, the child is joined to the same WLM enclave. This allows WLM to manage the calling process and child as one business unit of work entity for system accounting and management purposes.
9. z/OS UNIX sets a default message class of A for all forked, cloned, or spawned processes. Unlike JES, z/OS UNIX does not have a method for accepting a user-supplied default message class, and a default had to be supplied to the converter interpreter. Message class A was chosen as the default for BPXAS initiators. You cannot dynamically change this default value. The MSGCLASS for the job log (JESMSGLG, JESJCL, JESYSMSG) is set to class A before the fork or spawn that associates the process with the BPXAS initiator is begun.
10. The user syscall trace setting is propagated to the child process.

## close (BPX1CLO, BPX4CLO) — Close a file

---

### Function

The close callable service closes a file. You identify the file by its file descriptor.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB. If SRB, AF_INET/AF_INET6 socket support only
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CLO)</b>	31-bit task or SRB mode

Operation	Environment
<b>AMODE (BPX4CLO)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CLO, (File_descriptor,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CLO with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword containing the file descriptor of the file or socket the caller wants closed. The file descriptor is returned by the open service (see [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511) or by the socket service (see [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets ”](#) on page 865).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the close service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the close service stores the return code. The close service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The close service can return one of the following values in the Return\_code parameter:



Return_code	Explanation
<b>EAGAIN</b>	The service did not complete because the file descriptor specified is in use by another thread in the same process.
<b>EBADF</b>	The File_descriptor does not identify a valid, open file. The following reason codes can accompany the return code: JRCLNeedClose and JRNotForDir.
<b>EINTR</b>	The service was interrupted by a signal while it was processing the close request. The file may or may not be closed.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the close service stores the reason code. The close service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. Closing a file closes, or frees, the file descriptor by which the file was known to the process. The system can then reassign the file descriptor to the same file or to another file when it is opened.
2. Closing a file descriptor also unlocks all outstanding byte range locks that a process has on the associated file.
3. If a file has been opened by more than one process, each process has a file descriptor. When the last open file descriptor is closed, the file itself is closed. If the file's link count is zero at that time, the file's space is freed and the file becomes inaccessible. When the last open file descriptor for a pipe or FIFO special file is closed, any data remaining in the file is discarded.
4. The close callable service is for files or sockets.
5. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

**Related services**

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe” on page 551](#)
- [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets ” on page 865](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1CLO \(close\) example” on page 1229](#).

## **closedir (BPX1CLD, BPX4CLD) – Close a directory**

---

### **Function**

The closedir callable service closes a directory.

### **Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CLD):</b>	31-bit
<b>AMODE (BPX4CLD):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### **Format**

The syntax format is as follows:

```
CALL BPX1CLD,(Directory_file_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CLD with the same parameters.

### **Parameters**

#### **Directory\_file\_descriptor**

Supplied parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened.

#### **Return\_value**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the closedir service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the closedir service stores the return code. The closedir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The closedir service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The Directory_file_descriptor parameter does not represent an open directory.
<b>EINTR</b>	The service was interrupted by a signal while it was processing a closedir request. The directory may or may not be closed.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the closedir service stores the reason code. The closedir service returns Reason\_code only if Return\_value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory” on page 523](#)
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory ” on page 656](#)
- [“rewinddir \(BPX1RWD, BPX4RWD\) — Reposition a directory stream to the beginning ” on page 701](#)

**Characteristics and restrictions**

None.

**Examples**See [“BPX1CLD \(closedir\) example” on page 1229](#).

## cond\_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events

---

**Function**

The cond\_cancel callable service allows the thread to cancel the effects of a call to the cond\_setup service (BPX1CSE).

**Requirements****Condition****Authorization:****Environment**

Supervisor state or problem state, any PSW key

Condition	Environment
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CCA):</b>	31-bit
<b>AMODE (BPX4CCA):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CCA, (Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CCA with the same parameters.

## Parameters

### Return\_Value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service returns a 0 to indicate that the interest in event notifications has been canceled, or -1 if it has not.

### Return\_Code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service stores the return code. The cond\_cancel service stores a return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

### Reason\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service routine stores the reason code. The reason code further qualifies the return code value. The cond\_cancel service stores a reason code only when the return

value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. A program can use the cond\_cancel service to clean up when it uses the cond\_setup service, but does not call cond\_wait or cond\_timed\_wait. The cond\_setup service causes the thread to be eligible to receive event notifications. If the program running on the thread is no longer interested in these events, it should call cond\_cancel to tell the system that event notifications are no longer required.
2. If you intend to call cond\_wait or cond\_timed\_wait later to wait until some event occurs, use the cond\_setup service to make your program eligible to receive event notifications. The system notes that your program will be waiting for some other thread, either to send it a signal or to use the cond\_post service to send an event notification. Both of these require the use of z/OS UNIX services. If z/OS UNIX determines that it has become impossible to send a signal or event notification to your program, it checks to see whether your program is or will be calling the cond\_wait or cond\_timed\_wait services. If so, z/OS UNIX abnormally terminates your program to prevent it from waiting for something that cannot occur. For this reason, if your program uses the cond\_setup service but does not subsequently call either cond\_wait or cond\_timed\_wait, it should use the cond\_cancel service to cancel the setup to receive event notifications.
3. When the program cannot determine whether cond\_wait or cond\_timed\_wait has been called, it should call cond\_cancel to ensure that the thread is not eligible to receive event notifications.

## Related services

- [“cond\\_setup \(BPX1CSE, BPX4CSE\) — Set up to receive event notifications ” on page 130](#)
- [“cond\\_timed\\_wait \(BPX1CTW, BPX4CTW\) — Suspend a thread for a limited time or an event ” on page 132](#)
- [“cond\\_wait \(BPX1CWA, BPX4CWA\) — Suspend a thread for an event ” on page 136](#)

## Characteristics and restrictions

None.

## Examples

See [“BPXICCA \(cond\\_cancel\) example” on page 1225](#).

# cond\_post (BPX1CPO, BPX4CPO) — Post a thread for an event

---

## Function

The cond\_post callable service notifies another thread in the process that an event has occurred.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CPO):</b>	31-bit
<b>AMODE (BPX4CPO):</b>	64-bit
<b>ASC mode:</b>	Primary mode

**Operation****Environment****Interrupt status:**

Enabled for interrupts

**Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1CPO, (Thread_ID,  
               Event,  
               Return_value,  
               Return_code,  
               Reason_code)
```

AMODE 64 callers use BPX4CPO with the same parameters.

**Parameters****Thread\_ID**

Supplied parameter

**Type:**

Character string

**Length:**

8 bytes

The name of an 8-byte field that contains the thread ID for the thread that is to be notified of the event. The target thread must be in the same process as the caller.

**Event**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains an integer value that determines which event notification is to be sent to the target thread. The Event value represents an event for which the thread identified by Thread\_ID might be waiting. If the target thread is waiting, the cond\_post service notifies it that the event has occurred.

The value that is specified by Event must be one of the following two event values, which are defined by the BPXYCW macro:

- CW\_CONDVAR causes the target thread to resume processing if it is waiting for a CW\_CONDVAR event.
- CW\_TIMEOUT causes the target thread to resume processing if it is waiting for a timeout notification.

**Notes:**

1. You must specify exactly one event.
2. Use of cond\_post to send a CW\_TIMEOUT notification is restricted to programs that run in supervisor state with protect key 0.

**Return\_Value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service returns a 0 if an event notification was sent to the target thread, or -1 if it was not.

**Return\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service stores the return code. This service routine returns the return code only if the return value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The cond\_post service may return one of the following values in the return code parameter:

Error	Explanation
<b>EINVAL</b>	The value that is specified by Thread_ID is not valid. Either the Event parameter contains an incorrect value, or Thread_ID contains a lightweight thread ID. The following reason codes unique to the cond_post call can accompany this return code: JRLightWeightThID, JRNoEvents, JRTimeoutNotAuth, JRTooMany, JRUndefEvents.
<b>ESRCH</b>	The system determined that the value that was specified by Thread_ID does not refer to a thread that currently exists in the caller's process. The following reason codes can accompany this return code: JRThreadNotFound, JRAAlreadyTerminated.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service routine stores the reason code. The Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

The cond\_post service attempts to send an event notification to the target thread. Event notifications are delivered to a target thread only when the thread is set up to receive them. If the target thread is not set up to receive it, the event notification is discarded. The cond\_post service does not check whether the target thread is set up to receive the event, so the cond\_post service can return a value of 0 even though the event notification was discarded. Therefore, if you use the cond\_wait and cond\_post services to synchronize threads, you must be certain that the target thread is set up for the wait or in the wait before you use cond\_post to send the notification.

**Related services**

- [“cond\\_timed\\_wait \(BPX1CTW, BPX4CTW\) — Suspend a thread for a limited time or an event ”](#) on page 132
- [“cond\\_wait \(BPX1CWA, BPX4CWA\) — Suspend a thread for an event ”](#) on page 136

## Characteristics and restrictions

The target thread must be in the same process as the caller.

### Examples

See “BPX1CPO (cond\_post) example” on page 1230.

## cond\_setup (BPX1CSE, BPX4CSE) – Set up to receive event notifications

---

### Function

The cond\_setup callable service makes the calling thread eligible to receive event notifications from other threads.

### Requirements

**Operation****Environment****Authorization:**

Supervisor state or problem state, any PSW key

**Dispatchable unit mode:**

Task

**Cross memory mode:**

PASN = HASN

**AMODE (BPX1CSE):**

31-bit

**AMODE (BPX4CSE):**

64-bit

**ASC mode:**

Primary address space control (ASC) mode

**Interrupt status:**

Enabled for interrupts

**Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1CSE, (Event_list,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CSE with the same parameters.

### Parameters

**Event\_list**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains a value that specifies which events are of interest to the thread. The value contained in Event\_list is the inclusive OR of one or more of the following event values, which are defined by the BPXYCW macro:



**CW\_INTRPT**

The program that is running on the thread needs to know about signals sent to the thread.

**CW\_CONDVAR**

The program that is running on the thread needs to suspend processing until some other thread uses the cond\_post service to send this thread a notification of a CW\_CONDVAR event.

You must specify at least one event; you can specify both.

**Return\_Value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service returns a 0 upon normal completion, or -1 otherwise.

**Return\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the cond\_setup call stores the return code. The cond\_setup call stores return code only if return value is -1. For a list of return code values, see [Return codes \(ernnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The cond\_setup call can return one of the following values in the return code parameter:

Error	Explanation
<b>EINVAL</b>	The system determined that the event list that was passed to the service is in error. The following reason codes unique to the cond_setup call can accompany the return code: JRAlreadySetup, JRNoEvents, JRUndefEvents.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service routine stores the reason code. The cond\_setup service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The effects of the cond\_setup request remain until the next service is requested. The cond\_setup service is intended to be used to set up for a subsequent call to cond\_wait or cond\_timed\_wait. If the program invokes other callable services between cond\_setup, on the one hand, and cond\_wait or cond\_timed\_wait, on the other hand, cond\_wait or cond\_timed\_wait may fail with a return value of -1, a reason code of EINVAL, and a reason code of JRNotSetup.

The only exception to this is the queue\_interrupt service. You can use the queue\_interrupt service to "put back" the last signal delivered to the signal interface routine.

2. If you use cond\_setup to specify the events that cause the thread to resume processing, you must repeat the setup before each call to cond\_wait or cond\_timed\_wait.

3. If you use cond\_setup with cond\_timed\_wait, do not specify the CW\_TIMEOUT condition on the call to cond\_setup. The cond\_timed\_wait service provides setup for the CW\_TIMEOUT event.
4. Calling the cond\_setup service before the cond\_wait and cond\_timed\_wait services is optional. If the thread does not need to do any additional processing between the time it becomes eligible to request event notification and the time it suspends, you can specify the events on cond\_wait or cond\_timed\_wait instead of using cond\_setup.
5. If a thread has called cond\_setup but has not called cond\_wait or cond\_timed\_wait, any cond\_post services to it are remembered, and processed following the setup. When the following symbolic conditions are possible: or cond\_timed\_wait service is called, the pending cond\_post prevents the caller from waiting.

## Related services

- [“cond\\_cancel \(BPX1CCA, BPX4CCA\) — Cancel interest in events” on page 125](#)
- [“cond\\_post \(BPX1CPO, BPX4CPO\) — Post a thread for an event ” on page 127](#)
- [“cond\\_timed\\_wait \(BPX1CTW, BPX4CTW\) — Suspend a thread for a limited time or an event ” on page 132](#)
- [“cond\\_wait \(BPX1CWA, BPX4CWA\) — Suspend a thread for an event ” on page 136](#)
- [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered” on page 648](#)

## Characteristics and restrictions

The program running on the thread should eventually call one of the cond\_wait, cond\_timed\_wait, or cond\_cancel services.

## Examples

See [“BPX1CSE \(cond\\_setup\) example” on page 1231](#).

# cond\_timed\_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event

---

## Function

The cond\_timed\_wait callable service suspends the calling thread until any one of a set of events has occurred, or until a specified amount of time has passed.

## Requirements

Operation	Environment
<b>Authorization</b>	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CTW)</b>	31-bit
<b>AMODE (BPX4CTW)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1CTW, (Seconds,
               Nanoseconds,
               Event_list,
               Seconds_remaining,
               Nanoseconds_remaining,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CTW with the same parameters.

**Parameters****Seconds**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains an unsigned integer that is the maximum number of seconds that the calling program is willing to wait for one of the specified events to occur.

**Notes:**

1. Seconds can be any value greater than or equal to 0, and less than or equal to 4 294 967 295. The value that is specified for Seconds is an unsigned integer.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

**Nanoseconds**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains an unsigned integer that is the number of nanoseconds to be added to the value specified by Seconds.

**Notes:**

1. Nanoseconds can be any value greater than or equal to 0, and less than or equal to 1 000 000 000.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

**Event\_list**

Supplied parameter

**Type**

Integer

**Length**

Fullword

Event\_list specifies the name of a fullword that contains a value that determines which events are to cause the thread to resume processing.

## cond\_timed\_wait (BPX1CTW, BPX4CTW)

The value that is contained in the event list is the inclusive OR of one or more of the following event values, which are defined by the BPXYCW macro:

### CW\_INTRPT

Suspends processing until a signal is sent to the thread. This is a cancellation point that is described in the usage notes of [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state”](#) on page 610.

### CW\_CONDVAR

Suspends processing until some other thread in the process sends this one a CW\_CONDVAR notification.

If the event list is zero, the caller has used the cond\_setup service to specify the events, and the thread is already eligible to be notified of events. In this case, the cond\_timed\_wait service sets the timer for the specified interval, and suspends thread processing until an event occurs, a signal arrives, or the time limit is reached.

### Seconds\_remaining

Supplied returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the cond\_timed\_wait returns an unsigned integer that is the number of seconds of unexpired time remaining in the time interval.

**Note:** The Seconds\_remaining value is valid only when the return value is 0 or EINTR.

### Nanoseconds\_remaining

Supplied returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the cond\_timed\_wait returns an unsigned integer that is the number of nanoseconds of unexpired time remaining in the time interval.

#### Notes:

1. Nanoseconds\_remaining can be any value greater than or equal to 0, and less than or equal to 1 000 000 000.
2. The nanoseconds remaining value is valid only when the return value is 0 or EINTR.

### Return\_Value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the service returns a 0 if a CW\_CONDVAR event occurred, or -1 if it has not.

### Return\_Code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the service stores the return code. The cond\_timed\_wait service stores a return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* cond\_timed\_wait might return one of the following values in the Return\_code parameter:

Error	Explanation
<b>EAGAIN</b>	No signal or event notification arrived within the specified timeout period. The thread resumed processing because the time interval expired.  If you specify a value of zero for both Seconds and Nanoseconds, and no event notification is pending when you call cond_timed_wait, the service returns this error.
<b>EINTR</b>	A signal caused the cond_timed_wait service to resume processing of the thread.  The signal handler has already run.
<b>EINVAL</b>	The system determined that one or more of the parameters that were passed to the service are in error. The following reason codes unique to the cond_timed_wait call can accompany the return code: JRAlreadySetup, JRNanoSecondsTooBig, JRNotSetup, JRUndefEvents.

#### Reason\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the service routine stores the reason code. The cond\_timed\_wait service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The cond\_timed\_wait service is similar to the POSIX function nanosleep(). (Refer to the POSIX standard for a description of nanosleep().) If you need the nanosleep() function, you can use cond\_timed\_wait to implement your own version.
2. If your program uses cond\_timed\_wait to wait for events that it specified by calling cond\_setup, it must not call any other z/OS UNIX services between the calls to cond\_setup and cond\_timed\_wait. If the program invokes other callable services between cond\_setup and cond\_timed\_wait, the cond\_timed\_wait callable service fails with a return value of -1, a return code of EINVAL, and a reason code of JRNotSetup.  
  
The only exception to this is the queue\_interrupt service. You can use the queue\_interrupt service to "put back" the last signal delivered to the signal interface routine. A signal can arrive after the program that is running on the thread has called cond\_setup, and before it gets a chance to call cond\_timed\_wait. The program may choose to "put back" the signal to defer handling of it until a later time.
3. If you use cond\_setup to specify the events that are to cause the thread to resume processing, you must repeat the setup before each call to cond\_wait or cond\_timed\_wait.
4. If the caller has a PSW key of 0 or a key that is different from the one that was in effect when the process was created, cond\_timed\_wait gives a return value of -1 with a return code of EMVSERR and a reason code of JRPswKeyNotValid.
5. If the thread has been set up for signals, the cond\_timed\_wait service must run on the same request block (RB) that was used when the setup for signals was performed.

6. If you do not include the CW\_INTRPT event when you use cond\_timed\_wait, some services that are used by other threads or processes cannot cause the waiting thread to resume processing. In particular, the following services do not cause an event notification unless CW\_INTRPT is specified in the event list:

- kill
- pthread\_cancel
- pthread\_kill
- pthread\_quiesce

## Related services

- [“cond\\_cancel \(BPX1CCA, BPX4CCA\) — Cancel interest in events” on page 125](#)
- [“cond\\_post \(BPX1CPO, BPX4CPO\) — Post a thread for an event ” on page 127](#)
- [“cond\\_setup \(BPX1CSE, BPX4CSE\) — Set up to receive event notifications ” on page 130](#)
- [“cond\\_wait \(BPX1CWA, BPX4CWA\) — Suspend a thread for an event ” on page 136](#)
- [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered” on page 648](#)

## Characteristics and restrictions

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413.](#)

## Examples

See [“BPX1CTW \(cond\\_timed\\_wait\) example” on page 1231.](#)

# cond\_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event

## Function

The cond\_wait callable service allows the caller's thread to suspend processing until any one of a set of events has occurred.

## Requirements

### Operation

#### Authorization:

#### Dispatchable unit mode:

#### Cross memory mode:

#### AMODE (BPX1CWA):

#### AMODE (BPX4CWA):

#### ASC mode:

#### Interrupt status:

#### Locks:

#### Control parameters:

### Environment

Problem program or supervisor state, PSW key when the process was created (not PSW key 0)

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1CWA, (Event_list,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4CWA with the same parameters.

**Parameters**

**Event\_list**

Supplied returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains a value that determines which events will cause the thread to resume processing.

The value contained in Event\_list is the inclusive OR of one or more of the following event values defined by the BPXYCW macro.

**CW\_INTRPT**

Suspends processing until a signal is sent to the thread.

**CW\_CONDVAR**

Suspends processing until some other thread in the process sends this one a CW\_CONDVAR event notification.

An Event\_list of zero means that the caller has used the cond\_setup service to specify the events, and the thread is already eligible to be notified of events. In this case, the cond\_wait service suspends thread processing until an event occurs or a signal arrives.

**Return\_Value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service returns a 0 a CW\_CONDVAR event occurred, or -1 otherwise.

**Return\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service stores the return code. The cond\_wait service stores a return code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes* The cond\_wait service may return one of the following values in the Return\_code parameter:

Error	Explanation
<b>EINTR</b>	A signal caused the cond_wait service to resume processing of the thread. The signal handler has already run.

**Error****Explanation****EINVAL**

The system determined that one or more of the parameters that were passed to the service are in error. The following reason codes unique to the cond\_wait call can accompany the return code: JRAlreadySetup, JRNotSetup, JRUndefEvents.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service routine stores the reason code. The cond\_wait service stores a reason code only when the return value is =1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. If your program uses cond\_wait to wait for events that it specified by calling cond\_setup, it must not call any other z/OS UNIX services between the calls to cond\_setup and cond\_wait. If the program invokes other callable services between cond\_setup and cond\_wait, the cond\_wait callable service fails with a return value of -1, a return code of EINVAL, and a reason code of JRNotSetup.

The only exception to this is the queue\_interrupt service. You may use the queue\_interrupt service to "put back" the last signal delivered to the signal interface routine. A signal may arrive after the program that is running on the thread has called cond\_setup and before it gets a chance to call cond\_wait. The program may choose to "put back" the signal to defer handling it until a later time.

If you use cond\_setup to specify the events that will cause the thread to resume processing, you must repeat the setup before each call to cond\_wait or cond\_timed\_wait.

2. If the caller has a PSW key of 0 or a key that is different from the one that was in effect when the process was created, cond\_wait gives a return value of -1, a return code of EMVSERR, and a reason code of JRPswKeyNotValid.
3. If the thread has been set up for signals, the cond\_timed\_wait service must run on the same request block (RB) that was used when the setup for signals was performed.
4. If you do not include the CW\_INTRPT event when you use cond\_wait, some services that are used by other threads or processes cannot cause the waiting thread to resume processing. In particular, the following services do not cause an event notification unless CW\_INTRPT is specified in the event list:
  - kill
  - pthread\_cancel
  - pthread\_kill
  - pthread\_quiesce

**Related services**

- [“cond\\_cancel \(BPX1CCA, BPX4CCA\) — Cancel interest in events” on page 125](#)
- [“cond\\_post \(BPX1CPO, BPX4CPO\) — Post a thread for an event ” on page 127](#)
- [“cond\\_setup \(BPX1CSE, BPX4CSE\) — Set up to receive event notifications ” on page 130](#)
- [“cond\\_timed\\_wait \(BPX1CTW, BPX4CTW\) — Suspend a thread for a limited time or an event ” on page 132](#)
- [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered” on page 648](#)



## Characteristics and restrictions

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,”](#) on page 1413.

### Examples

See [“BPX1CWA \(cond\\_wait\) example”](#) on page 1231.

## connect (BPX1CON, BPX4CON) – Establish a connection between two sockets

---

### Function

For stream sockets, the connect callable service establishes a connection from a client socket to a socket at a server. For UDP (Universal Datagram Protocol) sockets, the connect callable service specifies the peer for a socket.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CON):</b>	31-bit task or SRB mode
<b>AMODE (BPX41CON):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1CON, (Socket_descriptor,
               Sockaddr_length,
               Sockaddr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CON with the same parameters.

### Parameters

#### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the connect is to be done.

**Sockaddr\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length of Sockaddr.

**Sockaddr**

Supplied parameter

**Type:**

Character

**Length:**

Length specified by Sockaddr\_length.

The name of a field that contains the address of the socket or the name of the peer to which a connection is to be attempted.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the connect service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the connect service stores the return code. The connect service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The connect service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	For AF_UNIX sockets, search permission is denied for a component of the path prefix, or write access to the named socket is denied.
<b>EAFNOSUPPORT</b>	The address family that was specified in the address structure is not supported.
<b>EBADF</b>	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNREFUSED</b>	The attempt to connect was rejected. The connect request might exceed the backlog count of the target socket, or the target socket might be closed. The following reason codes can accompany the return code: JRSocketNotFound, JRExceedsBacklogCount, JRListenNotDone.

Return_code	Explanation
<b>EINVAL</b>	The length that is specified in the Sockaddr_length or in the name length field in the Sockaddr is not valid. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoName.
<b>EINTR</b>	A signal interrupted the connect service before this connection was accepted. The following reason code can accompany the return code: JRSignalReceived.
<b>EIO</b>	There has been a network or transport failure. The following reason codes can accompany the return code: JRPrevSockError, JRTransportDriverNotAccessible.
<b>EINPROGRESS</b>	The socket is marked nonblocking and the connection cannot be completed immediately.
<b>EISCONN</b>	The socket is already connected.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EOPNOTSUPP</b>	The socket is ready to accept connections. An accept request was expected. The following reason code can accompany the return code: JRListenAlreadyDone.
<b>EPROTOTYPE</b>	The address specifies a socket that is not the correct type for this request. The following reason code can accompany the return code: JRIncorrectSocketType.
<b>EWouldBlock</b>	The socket is marked nonblocking, and the connection cannot be completed immediately.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the connect service stores the reason code. The connect service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. For connectionless sockets, the connect service might be advantageous because the destination address need not be specified for every datagram sent. Once a UDP (connectionless) socket is connected, the read, write, recv, and send system calls can be used for I/O on those sockets. Otherwise, only the sendto/recvfrom system calls can be used. Once a UDP socket is connected, only datagrams from the specified sockaddr are received on the socket. To disconnect a UDP socket from a previous connection, issue the connect system call with an invalid (null) sockaddr.
2. The connect callable service can be used to test whether a target socket is available for the connect. If the socket is not available, an ECONNREFUSED is returned.
3. The connect callable service will always either immediately succeed or fail, depending on the condition of the queue of pending connections, or backlog queue. If the backlog queue is not full, the connect request will immediately succeed. If the backlog queue is full, the connect request will fail with Return\_code of ECONNREFUSED.

4. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

## Characteristics and restrictions

None.

## Examples

See [“BPX1CON \(connect\) example” on page 1229](#).

# **\_\_console() (BPX1CCS, BPX4CCS) – Communicate with the console**

## Function

The `__console()` service sends messages to the console and waits on a modify/stop request from the console. Additional functions available under `__console2()` allow you to specify routing and descriptor codes for messages sent to the console and delete held messages from the console, using message IDs or tokens. These functions are activated under the expanded BPXYCCA structure in the Version 2 section. See the usage notes for information about using the `__console2()` functions. Additionally, the BPXYCCA structure Version 3 section provides the ability to specify a user-supplied CART and console ID. This function can be used by applications that communicate with more than one console to ensure that the messages go to the expected console.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1CCS):</b>	31-bit
<b>AMODE (BPX4CCS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax is as follows:

```
CALL BPX1CCS,(MsgAttributes_length,
              MsgAttributes,
              Modify_buffer_ptr,
              Modify_string_length,
              Console_command,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CCS with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

**Parameters****MsgAttributes\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the area that contains the message attributes of the message that is to be sent to the console. If the length is zero, the MsgAttributes parameter is ignored, and no message is sent to the console.

**MsgAttributes**

Supplied parameter

**Type:**

Structure

**Length:**

Specified by the MsgAttributes\_length parameter.

The name of the area that contains the message attributes of the message that is to be sent to the console. Included in this macro mapping are the address and length of the message to be sent. The area is mapped by BPXYCCA. For information about the content of this area, see [“BPXYCCA — Map input/output structure for \\_\\_console\(\)” on page 1060](#).

**Modify\_buffer\_ptr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

A fullword (doubleword) field that contains the address of a 128-byte buffer that is to be used to receive a string of EBCDIC data from the console modify command. All characters that appear to the right of the APPL= are placed into this buffer, left-aligned. The length of the string copied is returned in the Modify\_string\_length parameter. The data returned is folded to uppercase. If this parameter is zero, this service does not wait for or process any console modify/stop commands.

**Modify\_string\_length**

Returned parameter

**Type:**

Integer

**Character set:**

No restriction

**Length:**

Fullword

The name of a fullword in which the \_\_console() service returns the length of the modify string that is returned at the location that is specified by Modify\_buffer\_ptr. If the Modify\_buffer\_ptr is zero, this parameter is unchanged.

**Console\_command**

Returned parameter

**Type:**

Integer

**Character set:**

No restriction

**Length:**

Fullword

The name of a fullword in which the \_\_console() service returns the type of command that was issued. The values are CONSOLE\_MODIFY and CONSOLE\_STOP.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_console() service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_console() service stores the return code. The \_\_console() service returns Return\_code only if Return\_value is 0. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_console() service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	A message attribute was not valid. An error was detected in one of the fields described by BPXYCCA (JrMsgLength, JrMsgMaxLines, or JrMsgAttrErr).
<b>EINTR</b>	The syscall was interrupted by a signal.
<b>EFAULT</b>	User storage that was passed in could not be accessed. The reason code identifies the bad user storage (JrMsgIdList, JrDescList, JrRoutingList, and JrMsgId).
<b>EPERM</b>	The specified routing code requires the user to have superuser authority (JrAuthRoutingCode).

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_console() service stores the reason code. The \_\_console() service returns Reason\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Only one thread per address space is allowed to wait on console commands. If the Modify\_buffer\_ptr is nonzero, there can be no other instance of the \_\_console() service waiting for console input. This restriction applies to both the multithread and the multiprocess models. Subsequent attempts fail with an EMVSERR and JrNoMulti.

2. Messages sent to the console go to the last console that issued a modify command to this job. If no modify has been issued to this job, the message goes to the console that started this job. If this job was not started (that is, invoker created by the fork service), the message goes to the default console route code.

Routing codes specified in the message attribute area override the current message routing.

3. An invoker is deemed to have appropriate privileges for this service if the BPX.CONSOLE resource in the FACILITY class is defined and the invoker is permitted to that resource, or if the invoker is running with an effective UID of 0.
4. If the invoker does not have appropriate privileges (see “Authorization” on page 8), a message ID (BPXM023I) and the invoker's login name are prefixed to the specified message text. If the invoker has appropriate privileges, the invoker is responsible for its own message headers. Any message sent to the console should comply with MVS message guidelines. For more information about how to prefix messages with the correct message header, see *Message directory in z/OS MVS System Messages, Vol 3 (ASB-BPX)*. These guidelines are not enforced by this service.
5. The length of the message must be between 1 and 17850 characters for invokers with appropriate privileges, and between 1 and 17780 for invokers without appropriate privileges. The number of lines written to the console is limited to 255. In the case of an unprivileged user, one of those lines is used for the message ID and the invoker's login name. If the message length is exceeded, no lines are written and the service returns an EINVAL. If the number of lines is exceeded, the service returns an EINVAL, but the first 255 lines are written to the console.
6. The \_\_console() service provides limited formatting in that it recognizes the NEWLINE character and attempts to break on word boundaries. If a blank is found within the last 10 characters of the line, the \_\_console() service breaks the line there. If no blanks are found within the last 10 characters, the line break occurs after the 70th character.
7. Use of QEDIT and console service control blocks to listen to console commands, in combination with this service, may result in failures of EMVSERR JrUnexpectedErr.
8. Although the modify string buffer is 128 bytes, the maximum modify string that can be received from the console is less. The largest string that can be typed in from the console is 126 bytes, and this must include the modify command, job name, and APPL= parameters. For example, F SERVER01,APPL= consumes 16 characters of the 126-character string.
9. If the modify\_buffer\_ptr is specified, the invoking thread waits until either a modify command is issued to this thread's job, or a caught or terminating signal is generated to this thread. The \_\_console() service is also an interrupt point for pthread\_cancel.
10. If the console operator enters nothing after the APPL=, the Modify\_buffer is unchanged and a Modify\_string\_length of zero is returned.
11. If the Console\_command type returned is CONSOLE\_STOP, the Modify\_string\_length is set to zero. Console stop commands do not pass string data. It is up to the application to handle the stop command; the system takes no action against the process in response to a stop command. The application may choose to ignore the stop command, or terminate the process through services such as BPX1EXI (BPX4EXI).
12. To use the functions available under console2(), specify the new version (CCA\_#Ver02) and the correct length (CCA#Ver2Len) in the CCA when invoking the \_\_console() service.
13. The three \_\_console() operations (WTO, DOM, and WAIT) can be performed in a single request. The order of operations is WTO (issue messages), DOM (delete messages), and WAIT (for a MODIFY or STOP command).

## Characteristics and restrictions

None.

## Examples

See “BPX1CCS (\_\_console()) example” on page 1225.

## \_\_cpl (BPX1CPL) – CPL interface service

---

### Function

The \_\_cpl callable service calculates coupling facility structure sizes that are required by the CFRM (Coupling Facility Resource Manager) policy through a Web interface.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Problem program or supervisor state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE</b>	31-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1CPL,(FunctionCode,
               Bufferlen,
               Buffer,
               Return_value,
               Return_code,
               Reason_code)
```

### Parameters

#### FunctionCode

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value indicating the type of CPL function that was requested. Supported values are as follows:

- 1** Request data from available coupling facilities.
- 2** Request a structure size.  
ioshm0400i
- 3** Request a structure size and CFLevel.
- 4** Request a structure size and CFLevel with variable-length parameter list.



- 5** Request a structure size and CFLevel with IXLCSP parameter list.
- 6** Request data from available coupling facilities with IXLMG parameter list.
- 7** Request data from the CFRM policy with IXCQUERY parameter list.

These values are defined in \_\_cpl.h.

### Bufferlen

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the input length of the buffer.

### Buffer

Supplied parameter

#### Type:

Structure

#### Length:

Length specified by Bufferlen

The name of a fullword that represents the buffer in which the \_\_cpl service receives the input parameters from the Web and returns the results of the call.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the \_\_cpl service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the \_\_cpl service stores the return code. The \_\_cpl service stores a return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_cpl service may return the following values in the Return\_code parameter:

Return code	Explanation
<b>EMVSCPLERROR</b>	A __cpl service request failed. Consult reason_code, which will contain the reason code from the failing z/OS service, to determine the reason that the error occurred.
<b>EFAULT</b>	One of the parameters contained an address that was not accessible to the caller.

Return code	Explanation
<b>EINVAL</b>	The FunctionCode parameter contains a value that is not correct, or the input parameter list is built incorrectly. The following reason codes can accompany the return code: JRCPLInvFcnCode, JRCPLInvBuffLen, JRCPLBuffTooSmall, JRCPLInvStrucType, JRCPLFcnReq, JRCPLParmVer.
<b>EPERM</b>	The calling thread's address space is not permitted to the BPX.CF resource in the FACILITY class. The caller's address space must be permitted to the BPX.CF resource profile in the FACILITY class. The following reason code can accompany the return code: JRCPLNotAuth.
<b>ENOSYS</b>	The __cpl service request failed because the system is not at the correct level. The following reason code can accompany the return code: JRCPLCFNotFound.

**Reason\_code**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the \_\_cpl service stores the reason code. The \_\_cpl service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

There is no 64-bit version of the \_\_cpl callable service.

## Characteristics and restrictions

The \_\_cpl service is a privileged service; the caller must have read access to the BPX.CF resource profile in the FACILITY class.

# deletehfs (BPX1DEL, BPX4DEL) – Delete a program from storage

## Function

The deletehfs service deletes a previously loaded program from the storage of the caller's process.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1DEL)</b>	31-bit
<b>AMODE (BPX4DEL)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation**

**Environment**

**Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1DEL,(Entrypt_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4DEL with the same parameters. The Entrypt\_address parameter is a doubleword.

**Parameters**

**Entrypt\_address**

Supplied parameter

**Type**

Integer

**Length**

Fullword (doubleword)

A fullword (doubleword) pointer field that contains an entry point address that was returned by the loadhfs service for a z/OS UNIX program that was loaded into the caller's process.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the deletehfs service returns -1 if it is not successful. If it is successful, the deletehfs service returns zero.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the deletehfs service stores the return code. The deletehfs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The deletehfs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EINVAL	The entrypt_address parameter contains an entry point address that is not valid. The entry point address does not represent a currently loaded program in the caller's process.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the deletehfs service stores the reason code. The deletehfs service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. A call to BPX1DEL (BPX4DEL) to delete a program from storage may not actually cause the program to be removed from storage. If the program has been loaded more than once, the program remains in storage until BPX1DEL (BPX4DEL) has been called the same number of times that the program was loaded.
2. If a program that is loaded into storage with the loadhfs service is not deleted from storage, the program remains in storage until the calling task terminates, if it is not a pthread. If the caller is a pthread, the program remains in storage until the Initial Pthread Creating Task (IPT) terminates,
3. When the calling process is being debugged via the Ptrace service, a call to the deletehfs service generates a WastStopFlagDelete Ptrace event to the debugger process.

**Related services**

- [“loadhfs \(BPX1LOD, BPX4LOD\) — Load a program into storage by path name”](#) on page 393

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1DEL \(deleteHFS\) example”](#) on page 1232.

## **epoll\_create (BPX1EPC, BPX4EPC) — Open an epoll file descriptor**

**Function**

The epoll\_create service creates a new epoll instance and returns a file descriptor that refers to the epoll instance. The returned epoll instance can then be used to register interest in particular file descriptors that use the epoll\_ctl (BPX1EPL) service.

For the macro, see [“BPXYEPOL— Map epoll syscall parameters”](#) on page 1075.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1EPC)</b>	31-bit
<b>AMODE (BPX4EPC)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts

<b>Operation</b>	<b>Environment</b>
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1EPC,(Options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4EPC with the same parameters.

## Parameters

### Options

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that indicates the initialization options. This field is mapped by the BPXYEPOL macro. The values for this field are zero or:

Flag	Value	Description
<b>EPOLL_CLOEXEC</b>	1	Sets the epoll file descriptor to close-on-exec (FCTLCLOEXEC).

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the `epoll_create` service returns the **epoll** file descriptor that was created, or -1 if it is not successful.

### Return\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the **epoll\_create** service stores the return code. The **epoll\_create** service sets the `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\) in z/OS UNIX System Services Messages and Codes](#)

The **epoll\_create** service can return one of the following values in the `Return_code` parameter:

Return_code	Explanation
<b>EINVAL</b>	An invalid option was specified for <code>epoll_create</code> .
<b>ENFILE</b>	The system could not obtain a cell from the OFTE cell pool.

Return_code	Explanation
ENFILE	Out of nonexpandable cell pool cells.

#### Reason\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the `epoll_create` service stores the reason code. The `epoll_create` service sets the `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_value`. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### Usage notes

1. `EPOCH_CLOEXEC` indicates that the created `epoll` file instance is to be marked `FCTLCLOEXEC`. For more information, see [“BPXYFCTL — Command values and flags for `fcntl`”](#) on page 1076.

### Related services

- [“`epoll\_ctl` \(BPX1EPL, BPX4EPL\) — Control interface for `epoll` file descriptor”](#) on page 152
- [“`epoll\_wait` \(BPX1EPW, BPX4EPW\) — Wait for an I/O event on an `epoll` file descriptor”](#) on page 157

### Characteristics and restrictions

None.

### Examples

See [“BPX1EPC \(`epoll\_create`\) example”](#) on page 1233 and [“BPX4EPC \(`epoll\_create`\) example”](#) on page 1326.

## epoll\_ctl (BPX1EPL, BPX4EPL) — Control interface for `epoll` file descriptor

---

### Function

The **`epoll_ctl`** service is used to add, modify, or remove file descriptor entries in the interest list of the `epoll` instance. The I/O events may be registered for the specified file descriptor. Use the file descriptor for character-special files, pipes, sockets, inotify files, `eventFd`, and other `epoll` instances. The ready list can be retrieved with the **`epoll_wait`** (BPX1EPW, BPX4EPW) service.

For the macro, see [“BPXYEPOL— Map `epoll` syscall parameters”](#) on page 1075.

### Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1EPL)	31-bit

<b>Operation</b>	<b>Environment</b>
<b>AMODE (BP4EPL)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1EPL,( Epoll_File_Descriptor,
               Option,
               File_Descriptor,
               Epoll_Events_Length,
               Epoll_Events,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4EPL with the same parameters.

## Parameters

### Epoll\_File\_Descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that contains the file descriptor of an epoll instance.

### Options

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that indicates the actions to be performed on the provided `Epoll_File_Descriptor`. This field is mapped by the `BPXYEPOL` macro. The valid options for this field are:

Flag	Value	Description
<b>EPOLL_CTL_ADD</b>	0	Adds an entry to the interest list of the <code>Epoll_File_Descriptor</code> . The entry that is being added is described by <code>File_Descriptor</code> .
<b>EPOLL_CTL_MOD</b>	1	Modifies an entry in the interest list of the <code>Epoll_File_Descriptor</code> . The entry that is being modified is described by <code>File_Descriptor</code> .

Flag	Value	Description
<b>EPOLL_CTL_DEL</b>	2	Deletes an entry from the interest list of the Epoll_File_Descriptor. The entry that is being deleted is described by File_Descriptor.  The events parameter is ignored when this option is specified.

**File\_Descriptor**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword field that contains the file descriptor corresponding to an open file description.

**Epoll\_Events\_Length**

Supplied parameter.

**Type**

Structure

**Length**

Fullword

The name of a fullword field that contains the length of the Epoll\_Events area. To determine the value of Epoll\_Events\_Length, use the BPXYEPOL macro.

**Epoll\_Events**

Supplied parameter.

**Type**

Structure

**Length**

Specified by the Epoll\_Events\_Length parameter.

The name of an area of length Epoll\_Events that contains the events for the File\_Descriptor. The BPXYEPOL macro maps this section.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

**epoll\_ctl** returns 0 on success and -1 on failure. It does not create an epoll file descriptor.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **epoll\_ctl** service stores the return code. The **epoll\_ctl** service sets the Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The **epoll\_ctl** service can return one of the following values in the Return\_code parameter:



Return_code	Explanation
<b>EBADF</b>	A provided <code>file_descriptor</code> subparameter does not contain the descriptor of an open file.
<b>EEXIST</b>	The supplied file descriptor is already registered with the epoll file descriptor.
<b>EINVAL</b>	The provided epoll file descriptor and file descriptor to be monitored are the same.
<b>EINVAL</b>	The file descriptor that was passed in <i>argument 1</i> was not an epoll file descriptor.
<b>EINVAL</b>	The event cannot be identified as valid. The eye catcher value is incorrect.
<b>EINVAL</b>	An unsupported event was specified.
<b>EINVAL</b>	No events were specified.
<b>EINVAL</b>	The value that was passed in the epoll event length parameter is invalid.
<b>ENFILE</b>	EPOLLPRI or EPOLLONESHOT cannot be specified with EPOLLEXCLUSIVE.
<b>ENFILE</b>	An epoll file cannot be watched exclusively.
<b>EINFILE</b>	A file is either being modified or is already exclusively watched.
<b>EINVAL</b>	An option other than EPOLL_CTL_ADD, EPOLL_CTL_MOD, or EPOLL_CTL_DEL was specified for <b>epoll_ctl</b> .
<b>ENOENT</b>	The supplied file descriptor is not registered with this epoll file.
<b>EPERM</b>	The requested service is not supported for this file type.
<b>EMFILE</b>	The total number of file descriptors that a user can register across all epoll files on the system was reached. The limit is per real user ID.
<b>ENOMEM</b>	Out of nonexpandable cell pool cells.
<b>ENOSPC</b>	The per-user limit on how many File_Descriptors can be in the interest list was reached.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **epoll\_ctl** service stores the reason code. The **epoll\_ctl** service sets the Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The `Epoll_Events` area is mapped by BPXYEPOL. The `EPOLL_ID` area must be set to `EPOLLEV#ID`.
2. The `EPOLLEVENTS` area is a 32-bit mask field. EPOLLIN, EPOLLOUT, EPOLLPRI, or a combination of the three must be specified. The following events can be OR'ed together to specify the I/O events to be monitored by **epoll**.

Event	Description
EPOLLIN	The file is ready for read operations.

Event	Description
EPOLLOUT	The file is ready for write operations.
EOPOLLPRI	Out-of-band data can be received without blocking.
EPOLLERR	An error condition occurred on the associated file descriptor. This error is also reported if any side of a pipe is closed.  <b>epoll_wait</b> will always report this event. You do not have to request this as an I/O event.
EPOLLONESHOT	Requests that one-shot notification for the associated file descriptor. After an event is notified for the file descriptor by <b>epoll_wait</b> , the file descriptor is disabled in the interest list and no other events will be reported by the epoll interface.  The user must call <b>epoll_ctl</b> with EPOLL_CTL_MOD to rearm the file descriptor with a new event mask.  This is an input value for <b>epoll_ctl</b> . <b>epoll_wait</b> never returns this value.
EPOLLEXCLUSIVE	Sets an exclusive wake-up mode for the epoll file descriptor that is being attached to the target file descriptor (File_Descriptor). When a wake-up event occurs and multiple epoll file descriptors are attached to the same target file using EPOLLEXCLUSIVE, one or more of the epoll file descriptors will receive an event with <b>epoll_wait</b> . The default in this scenario (without EPOLLEXCLUSIVE), is for all epoll file descriptors to receive an event. EPOLLEXCLUSIVE avoids thundering herd problems.  If the same file descriptor is in multiple epoll instances, some with EPOLLEXCLUSIVE and others without, then events are provided to all epoll instances that did not specify EPOLLEXCLUSIVE, and at least 1 of EPOLLEXCLUSIVE instances.  EPOLLIN and EPOLLOUT can be specified with EPOLLEXCLUSIVE. Attempts to specify other values will yield EINVAL.  EPOLLEXCLUSIVE can be used only in an EPOLL_CTL_ADD option. Attempts to use EPOLL_CTL_MOD to specify EPOLLEXCLUSIVE or on a file that is already marked as such will yield an EINVAL.  EPOLLEXCLUSIVE is an input value for <b>epoll_ctl</b> . <b>epoll_wait</b> never returns this input value.

### Related services

- [“epoll\\_create \(BPX1EPC, BPX4EPC\) — Open an epoll file descriptor” on page 150](#)
- [“epoll\\_wait \(BPX1EPW, BPX4EPW\) — Wait for an I/O event on an epoll file descriptor” on page 157](#)

### Characteristics and restrictions

None.

### Examples

See [“BPX1EPL \(epoll\\_epl\) example” on page 1233](#) and [“BPX4EPL \(epoll\\_epl\) example” on page 1327](#).

# epoll\_wait (BPX1EPW, BPX4EPW) – Wait for an I/O event on an epoll file descriptor

## Function

The **epoll\_wait** service waits for the requested I/O events of an interest list that were previously registered with a provided epoll instance. This function returns the ready list of file descriptors.

For the macro, see [“BPXYEPOL— Map epoll syscall parameters” on page 1075](#).

## Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1EPW)	31-bit
AMODE (BPX4EPW)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1EPW, (Epoll_File_Descriptor,
               Max_Events,
               Timeout,
               Events_Length,
               Events,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4EPW with the same parameters.

## Parameters

### Epoll\_File\_Descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that contains the file descriptor of an epoll instance.

### Max\_Events

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that describes the maximum number of events that may be returned by the **epoll\_wait** service. This value must be greater than zero.

**Timeout**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword field that contains a timeout value in milliseconds that the **epoll\_wait** service will block. Valid values are as follows:

**0**

No waiting. If the timeout value is 0, **epoll\_wait** returns immediately with any file descriptors that are in the list, if there are any.

**-1**

Wait forever. If the timeout value is -1, **epoll\_wait** blocks until a file descriptor is in the ready list.

**n**

If the timeout value is  $n > 0$ , it specifies that the number of milliseconds to wait for a ready list to be populated before returning to the caller.

**Events\_Length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of the Events area. To determine the value of each Event, use the BPXYEPOL macro. The Events\_Length parameter should be a value equal to the multiple of Max\_Events and EPOLLEV#LEN.

**Events**

Supplied parameter.

**Type**

Area

**Length**

(Max\_Events \* Events structure Length)

The name of an area that contains the returned area. the ready list is populated in this area. The number of events returned, if any, is provided as Return\_value. The event structure is mapped by the BPXYEPOL macro.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **epoll\_wait** service returns the **epoll\_wait** service returns one of the following return values:

**-1**

The request was unsuccessful.

**0**

If no file descriptors were in the ready list when the timeout was met or if a signal was caught.

***n***

The number of file descriptors that are ready for the requested I/O.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **epoll\_wait** service stores the return code. The **epoll\_wait** service sets the Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\) in z/OS UNIX System Services Messages and Codes](#)

The **epoll\_wait** service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The Epoll_File_Descriptor does not contain the descriptor of an open file.
<b>EINTR</b>	The service was interrupted by a signal handler before any of the requested events occurred.
<b>EINVAL</b>	MAX_Events is empty.
<b>EINVAL</b>	The requested service is not supported for this file type.
<b>EINVAL</b>	The buffer for return information is too small.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **epoll\_wait** service stores the reason code. The **epoll\_wait** service sets the Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Related services**

- [“epoll\\_create \(BPX1EPC, BPX4EPC\) — Open an epoll file descriptor” on page 150](#)
- [“epoll\\_ctl \(BPX1EPL, BPX4EPL\) — Control interface for epoll file descriptor” on page 152](#)

**Characteristics and restrictions**

None.

**Examples**

[“BPX1EPW \(epoll\\_wait\) example” on page 1234](#) and [“BPX4EPW \(epoll\\_wait\) example” on page 1327](#).

## eventfd (BPX1EFD, BPX4EFD) – Create a file descriptor for event notification

---

### Function

The **eventfd** service creates a new **eventfd** instance and returns a file descriptor that refers to that instance.

For the macro, see [“BPXYEFD— Map eventfd syscall parameters” on page 1075](#).

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1EFD)</b>	31-bit
<b>AMODE (BPX4EFD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1EFD, (Initval,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4EFD with the same parameters.

### Parameters

#### Initval

Supplied parameter.

#### Type

Unsigned 32-bit integer

#### Length

Fullword

#### Flags

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that indicates the Initialization options. This field is mapped by the BPXYEFD macro.

Flag	Value	Description
<b>EFD_CLOEXEC</b>	'00001000' X	Sets the <b>eventfd</b> file descriptor to close-on-exec.
<b>EFD_NONBLOCK</b>	'00000004' X	Sets the O_NONBLOCK file status flag on the open file description referred to by the new file descriptor. For more information, see <a href="#">“Usage notes” on page 161</a> .
<b>EFD_SEMAPHORE</b>	'00002000' X	Provides semaphore-like semantics for reads from the new file descriptor. For more information, see <a href="#">“Usage notes” on page 161</a> .

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **eventfd** initialization service returns the **eventfd** file descriptor that was created, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **eventfd** service stores the return code. The **eventfd** service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\) in z/OS UNIX System Services Messages and Codes](#) A list of possible return codes follows.

Return_code	Explanation
<b>EINVAL</b>	The Flags parameter does not specify a valid value.
<b>EMFILE</b>	The process reached the maximum number of file descriptions that can be open.
<b>ENFILE</b>	The maximum number of file descriptions that can be open was reached.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **eventfd** service stores the reason code. The **eventfd** service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. The following operations can be performed on the file descriptor that is returned by **eventfd**:

**read**

Each successful read returns an 8-byte integer. A read fails with the error EINVAL if the size of the supplied buffer is less than 8 bytes.

The value that is returned by read is in host byte order. That is, it is in the native byte order for integers on the host machine.

The semantics of read depend on whether the **eventfd** counter has a nonzero value and whether the EFD\_SEMAPHORE flag was specified when the eventfd file descriptor was created.

- If EFD\_SEMAPHORE was not specified and the **eventfd** counter has a nonzero value, then a read returns 8 bytes containing that value. The value of the counter is reset to zero.
- If EFD\_SEMAPHORE was specified and the **eventfd** counter has a nonzero value, then a read returns 8 bytes containing the value 1 and the counter's value is decremented by 1.
- If the **eventfd** counter is zero at the time of the call to read, then the call either blocks until the counter becomes nonzero (at which time, the read continues) or fails with the error EAGAIN if the file descriptor was made nonblocking.

BPX1RDV is not supported.

**write**

A write call adds the 8-byte integer value that is supplied in its buffer to the counter. The maximum value that can be stored in the counter is the largest unsigned 64-bit value minus 1, for example, 0xfffffffffffffffe. If the addition will cause the value of the counter value to exceed the maximum, then the write either blocks until a read is performed on the file descriptor or fails with the error EAGAIN if the file descriptor was made nonblocking.

A write fails with the error EINVAL if the size of the supplied buffer is less than 8 bytes or if an attempt was made to write the value 0xfffffffffffffffe.

BPX1WRV is not supported.

**poll**

The returned file descriptor supports poll as follows:

- The file descriptor is readable if the counter has a value greater than 0.
- The file descriptor is writable if it is possible to write a value of at least 1 without blocking.
- If an overflow of the counter value was detected, then select indicates the file descriptor as being both readable and writable, and poll returns a POLLERR event. As previously noted, write can never overflow the counter.

**close**

Closes file descriptors. When the file descriptor is no longer required, it should be closed. When all file descriptors that are associated with the same eventfd object have been closed, the kernel frees the resources for object.

A copy of the file descriptor that was created by **eventfd** is inherited by the child that was produced by fork. The duplicate file descriptor is associated with the same **eventfd** object. File descriptors that are created by **eventfd** are preserved across **execve**, unless the close-on-exec flag was set.

**Related services**

- [“close \(BPX1CLO, BPX4CLO\) — Close a file” on page 121](#)
- [“poll \(BPX1POL, BPX4POL\) — Monitor activity on file descriptors and message queues” on page 564](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

**Characteristics and restrictions**

None.



## Examples

See “BPX1EFD (eventfd) example” on page 1232 and “BPX4EFD (eventfd) example” on page 1326.

## exec (BPX1EXC, BPX4EXC) – Run a program

---

### Function

The exec callable service runs a z/OS UNIX executable file that is either a program object or a REXX exec. The exec callable service replaces the current process image that calls the exec service with a new process image for the executable file that is being run.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1EXC)</b>	31-bit
<b>AMODE (BPX4EXC)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1EXC, (Pathname_length,
               Pathname,
               Argument_count,
               Argument_length_list,
               Argument_list,
               Environment_count,
               Environment_data_length,
               Environment_data_list,
               Exit_routine_address,
               Exit_parameter_list_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4EXC with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

### Parameters

#### Pathname\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Pathname parameter. The length can be up to 1023 bytes long.

**Pathname**

Supplied parameter

**Type**

Character string

**Character set**

No restriction

**Length**

Specified by the Pathname\_length parameter

The name of a field that contains the fully qualified path name of the file to be run. Each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

The path name can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory. The search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Argument\_count**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the number of pointers in the lists for the Argument\_length\_list and the Argument\_list. If the program needs no arguments, define Argument\_count as the name of a fullword that contains 0.

**Argument\_length\_list**

Supplied parameter

**Type**

Structure

**Length**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an argument that is to be passed to the specified program. If the program needs no arguments, define Argument\_length\_list as the name of a fullword (doubleword) that contains 0.

**Argument\_list**

Supplied parameter

**Type**

Structure

**Length**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an argument to be passed to the specified program. Each argument is of the length that is specified by the corresponding element in the Argument\_length\_list. If the program needs no arguments, define Argument\_list as the name of a fullword (doubleword) that contains 0.

If the target executable file arguments require null terminators, the arguments that are supplied to this service must include the null terminator as part of the data string and the length.

### **Environment\_count**

Supplied parameter

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword that contains the number of pointers in the lists for Environment\_data\_length and Environment\_data. If the program needs no environment data, define Environment\_count as the name of a fullword that contains 0.

### **Environment\_data\_length**

Supplied parameter

#### **Type**

Structure

#### **Length**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an environment variable to be passed to the specified program. If the program does not use environment variables, define Environment\_data\_length as the name of a fullword (doubleword) that contains 0.

### **Environment\_data\_list**

Supplied parameter

#### **Type**

Structure

#### **Length**

Variable as specified by Environment\_data\_length

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an environment variable to be passed to the specified program. Each environment variable is of the length that is specified by the corresponding element in Environment\_data\_length. If the program does not use environment variables, define Environment\_data\_list as the name of a fullword (doubleword) that contains 0. If the target executable file is a Language Environment-enabled program, the environment variables that are supplied to this service must include the null terminator as part of the data string and length.

### **Exit\_routine\_address**

Supplied parameter

#### **Type**

Address

#### **Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be called, define Exit\_routine\_address as the name of a fullword (doubleword) that contains 0.

### **Exit\_parameter\_list\_address**

Supplied parameter

#### **Type**

Address

#### **Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value that is contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be called or does not require parameters, define `Exit_parameter_link_address` as the name of a fullword (doubleword) that contains 0. Because the exit must be RMODE 31, the address must be below the 2-gigabyte bar.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the exec service returns -1 if it is not successful. If it is successful, the exec service does not return.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the exec service stores the return code. The exec service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The exec service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The caller does not have appropriate permissions to run the specified file. It might lack permission to search a directory that is named in the <code>Pathname</code> parameter. For example, it might lack execute permission for the file to be run. Or, the file to be run is not a regular file and the system cannot run files of its type. The following reason code can accompany the return code: <code>JRExecNotRegFile</code> .
<b>EFAULT</b>	A bad address was received as an argument of the call, or the user exit program checked. The following reason code can accompany the return code: <code>JRExecParmErr</code> and <code>JRExitRtnError</code> .
<b>ELOOP</b>	A loop exists in symbolic links that are encountered during resolution of the <code>Filename</code> argument. This error is issued if more than 24 symbolic links are detected in the resolution of <code>Filename</code> .
<b>EMVSSAF2ERR</b>	The executable file is a set-user-ID or set-group-ID file, and the file owner's UID or GID is not defined to RACF.
<b>ENAMETOOLONG</b>	<code>File_name</code> is longer than 1023 characters, or some component of the file name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file name was specified, or one or more of the components of the specified <code>Filename</code> were not found. The following reason codes can accompany the return code: <code>JRExecNmLenZero</code> and <code>JRQuiescing</code> .
<b>ENOEXEC</b>	The specified file has execute permission, but it is not in the proper format to be a process image. <code>Reason_code</code> contains the loader reason code for the error.
<b>ENOMEM</b>	The new process requires more memory than is permitted by the hardware or the operating system. The following reason code can accompany the return code: <code>JRExecFileTooBig</code> .

Return_code	Explanation
ENOTDIR	A directory component of Filename is not a directory.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the exec service stores the reason code. The exec service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see Reason codes in *z/OS UNIX System Services Messages and Codes*. For the ENOEXEC Return\_code, Reason\_code contains the loader reason code for the error:

Reason code	Explanation
X'xxxx0C27'	The target file is not in the correct format to be an executable file.
X'xxxx0C31'	The target file is built at a level that is higher than the level that is supported by the running system.

**Usage notes for exec**

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes **exec** to execute a z/OS UNIX executable program file, certain conditions must be met.

- The program file and any symbolic links that comprise the path name to the file must have the proper attributes to allow the invocation.
- The z/OS UNIX executable file must be installed with an owning UID of 0 or with an owning UID equal to that of the calling set-user-ID program. Otherwise, it must have the program control extended attribute.

Any symbolic link that comprises the path name to the file or any external symbolic link, must follow these conditions, except that the program control extended attribute cannot be used because links do not have extended attributes.

Additionally, a file or link found in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Not following this setup causes the caller of the exec service to end abnormally with a EC6-xxxxE04B abend. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes exec to run a nonprivileged program, the new process image that is created to run the program inherits the privilege of the set-user-ID or set-group-ID program.

If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes exec to execute a z/OS UNIX executable program file, the program file and any symbolic links that comprise the path name to the file must have the proper attributes to allow the invocation.

- The z/OS UNIX executable file must be installed with an owning UID of 0 or with an owning UID equal to that of the calling set-user-ID program or have the program control extended attribute.
- The same is true for any symbolic link that comprises the path name to the file or any external symbolic link, except that, the program control extended attribute cannot be used because links do not have extended attributes.

Additionally, a file or link found in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Not following this setup causes the caller of the exec service to end abnormally with a EC6-xxxxE04B abend. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes exec to run a non-privileged program, the new process image created to run the program inherits the privilege of the set-user-ID or set-group-ID program.

2. The following characteristics of the calling process are changed when the service gives control to the new executable file:

- The current process image is replaced with a new process image for the executable file to be run.
- All directories that are opened via a call to the opendir service are closed in the new process image.
- All open file descriptors remain open unless the close-on-exec flag is set.
- Signals set to be caught are reset to their default.

If the SSTFNOSUID bit is set for the file system that contains the new process image file, the effective user ID, effective group ID, saved set-user-ID and saved set-group-ID are unchanged in the new process image. Otherwise, if the setuid bit of the new process image file is set, the effective user ID of the new process image is set to the owner ID of the new process image file.

Similarly, if the setgid bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image is saved (as the saved set-user-ID and the saved set-group-ID) for use by the setuid and setgid functions. For more information, see [“BPXYMODE — Map the mode constants”](#) on page 1106.

If an exit was registered to get control during a shutdown by an unauthorized caller and the new process image will run authorized, then the exit address will not be propagated to the new process.

3. The executable file to be run receives control with the following attributes:

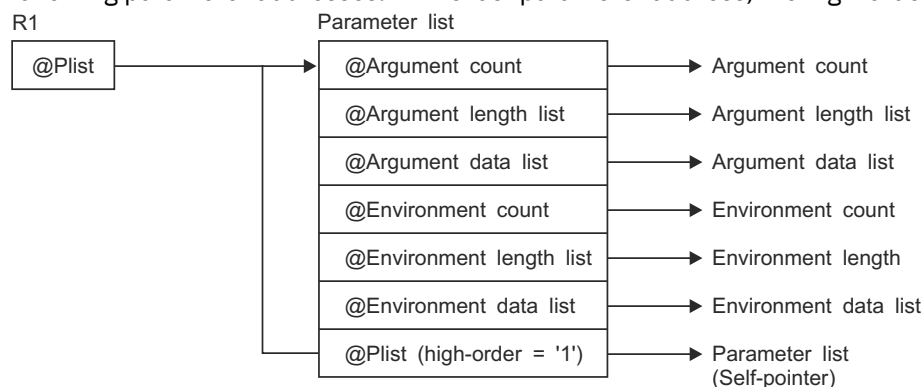
- Problem program state
- PSW key 8
- AMODE=31(64), taken from the executable
- Primary ASC mode

4. The new process image inherits the following from the calling process image:

- Process ID
- Parent process ID
- The time that is left until an alarm signal is generated
- File mode creation mask
- Process signal mask
- Pending signals
- Time accounting information

For more information, see [“times \(BPX1TIM, BPX4TIM\) — Get process and child process times”](#) on page 950 and [“BPXYTIMS — Map the response structure for times”](#) on page 1168.

5. The information that the service passes to the executable file to be run is a parameter list, which is pointed to by register 1. The parameter list consists of the following parameter addresses. In the last parameter address, the high-order bit is 1.



For AMODE 31 callers, the high-order bit in the last parameter address is 1. For AMODE 64 callers, the high-order bit is part of the 64-bit address. There are always  $n$  parameters, passed with no end-of-parameter-list indicator.

The last parameter that the exec service passed to the executable file identifies the caller of the file as the exec service. The exit gets control in the same AMODE as the caller.

6. The register usage on entry to the user exit in AMODE 31 is:

- R0: Undefined.
- R1: Address of the user exit parameter list, as specified by the caller of the exec service.
- R2–R12: Undefined.
- R13: Address of a 96-byte work area in the same key as the caller of the exec service.
- R14: The return address from the user exit to the exec service. This address must be preserved by the user exit.
- R15: Address of the user exit.

7. The register usage on entry to the user exit in AMODE 64 is:

- R0: Undefined.
- R1: 64-bit address of the user exit parameter list, as specified by the caller of the exec service.
- R2–R12: Undefined.
- R13: Address of a 96-byte work area in the same key as the caller of the exec service.
- R14: The return address from the user exit to the exec service. This address must be preserved by the user exit.
- R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.

8. When the exec or execmvs service is called in any environment except single task, single RB, and no linkage stack, z/OS UNIX issues an abend EC6. This takes down the calling task and all of its subtasks. The subtasks receive a 33E abend. All other thread tasks in the address space receive a 422 abend with a reason code of 00000181, and their subtasks receive a 33E abend.

9. To support the creation and propagation of a STEPLIB environment to the new process image, the exec service allows for the specification of a STEPLIB environment variable. The following table lists the accepted values for the STEPLIB environment variable and the actions taken for each value.

Value	Action taken
STEPLIB=NONE	No STEPLIB DD is created for the new process image.
STEPLIB=CURRENT	The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
STEPLIB=Dsn1:Dsn2:,...DsnN	<p>The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.</p> <p><b>Note:</b> The actual name of the DD is not STEPLIB. Instead, it is a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order specified. The specified data sets must follow standard MVS data set naming conventions. If they do not follow the standard naming conventions, they are ignored. The data sets are also ignored</p>

Value	Action taken
	<p>if they do follow the standard naming conventions but these situations exist:</p> <ul style="list-style-type: none"> <li>• The caller does not have the proper security access to a data set.</li> <li>• A data set is uncataloged or is not in load library format.</li> </ul> <p>The system will also ignore the data set and continue allocating the other data sets in these situations:</p> <ul style="list-style-type: none"> <li>• If the system cannot acquire a shared ENQueue on the data set name because another job has ENQueued it exclusively.</li> <li>• If the specified data set does not exist. This case is similar but not strictly equal to the case of uncataloged data sets.</li> </ul> <p>Because the data sets are ignored, the executable file might run without the proper STEPLIB environment. If a data set is ignored due to improper security access, an X'913' abend is generated. The dump for this abend can be suppressed by your installation</p> <p>If the program to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For more information about the sanction list, see <a href="#">Using sanction lists in z/OS UNIX System Services Planning</a>. For information about STEPLIB performance considerations, see <a href="#">Controlling use of STEPLIBs in z/OS UNIX System Services Planning</a>.</p>

If the STEPLIB environment variable is not specified, the default behavior of the exec service is the same as if STEPLIB=CURRENT were specified.

10. The `_BPX_JOBNAME` environment variable can be used to change the job name of the new process image. The job name change is allowed only if the invoker has appropriate privileges (see “Authorization” on page 8) and is running in an address space that is created by fork. If these conditions are not met, the environment variable is ignored. Accepted values are strings of 1–8 alphanumeric characters. Incorrect specifications are ignored.
11. The `_BPX_ACCT_DATA` environment variable can be used to change the account data of the new process image. Specifying this environment variable triggers a new job. The rules for specifying the account data are as follows:
  - Up to 142 actual account data characters are allowed, including any commas.
  - Subparameters must be separated by commas.
  - There is no restriction on the character set.

If the account data is greater than 142 characters, the data is ignored. No other validity or syntax checking will be done.
12. Each shared-memory segment attached to the calling process is detached, and the value of the number of processes that are attached to each detached segment (`shm_nattch`) is decremented by 1. If this is the last process that is attached to a shared memory segment and `shmctl` (BPX1SCT, BPX4SCT) `IPC_RMID` has been issued for the shared memory segment, the segment will be removed from the system.
13. The semaphore adjustment value, `semadj`, is inherited by the new process.



14. A prior loaded copy of a program in the same address space is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS XCTL service, with the following exceptions:
  - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
  - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the program found is in storage modifiable by the caller, the prior copy is not reused.
15. If the specified file or link resolves to a MVS program that is link-edited AC=1 located in an APF-authorized library, the attributes of the sticky bit file or external link must be set up properly to allow this type of invocation. For a sticky bit file, it must be installed with an owning UID of 0 or with the APF extended attribute. The owning UID of 0 requirement would also apply to a symbolic link that resolves to the sticky bit file. For an external link, it must be installed with an owning UID of 0. Also, a file with the APF extended attribute is not allowed if found in a file system mounted as NOSETUID. If the specified file name represents a symbolic link to a sticky bit file that has the set-user-id attribute, the symbolic link must have an owning uid of 0 or an owning UID equal to that of the sticky bit file. If the sticky bit file has the set-group-id attribute, the symbolic link must have an owning uid of 0 or an owning GID equal to that of the sticky bit file. A file or link found in a file system mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Failure to follow this set up will cause the executing job to end abnormally with a EC6-xxxxC04A abend when invoking the MVS program via the exec service.

The following restriction exists for sticky bit programs that have the set-user-ID or set-group-ID attributes:

- If the program is found in the MVS program search order, the MVS program name must have a BPX.STICKYSUG.*program\_name* resource profile defined in the RACF FACILITY class to be successfully exec'ed. For examples of BPX.STICKYSUG.*program\_name*, see [Setting up the UNIX-related FACILITY and SURROGAT class profiles in z/OS UNIX System Services Planning](#).

Failure to follow this restriction will cause the caller of exec to end abnormally with a EC6-xxxxE055 abend.

16. If the specified file is not in the correct format to be an executable, but contains the "magic number" (#!) in the first line, the program that is specified in the magic number header is executed. The expected format of the magic number header is as follows:

```
#! Path String
```

#! is the file magic number. It identifies the first line of the file as a special header that contains the name of the program to be run and any argument data to be supplied to it.

The **Path** parameter specifies the path name of the file that is to be run. It is separated by blank or tab characters from the #! characters, or can immediately follow the characters.

The **String** parameter is an optional character string that can be used to pass options to a target command interpreter (shell) that is to run the script. It must be separated from the **Path** parameter by tab or blank characters, and cannot itself contain tab or blank characters.

The argument data list and argument length list that are passed to the magic number file are to contain the following argument data and corresponding argument data lengths:

- The magic number path name, ended by a null character (X'00')
- The string, if one is supplied, ended by a null character (X'00')

The remaining arguments in the list are to contain the list of arguments that are specified by the caller of the exec service.

If the path name that is specified in the magic number header cannot be executed for some reason, the exec request fails with return code ENOEXEC, regardless of the error. ENOEXEC is returned for compatibility purposes, so that existing scripts can continue to run successfully when invoked from an application such as a command interpreter (shell). The reason code indicates the exact reason the magic number file could not be executed.

17. If the calling task is in a WLM enclave, the resulting task in the new process image is joined to the same WLM enclave. This allows WLM to manage the old and new process images as one business unit of work entity for purposes of system accounting and of management.
18. If the target executable program is an IBM Language Environment-enabled program, the environment variables that are supplied to the service must include the null terminator as part of the string and length.
19. If the program that is being executed is APF-authorized, link-edited with AC=1, and is being executed on the job step task, the address space is marked as APF-authorized.
20. If the \_BPX\_PTRACE\_ATTACH environment variable is set to YES, the target executable program is loaded into user-modifiable storage to allow subsequent debugging. Any additional programs loaded into storage during the execution of the target program are also loaded into user-modifiable storage, with the exception of modules loaded from the LPA.
21. The \_BPXK\_MDUMP environment variable can be used to specify where a SYSMDUMP is to be written. The following table lists the allowable values:

Value	Description
OFF	The dump is written to the current directory. This is the default. This dump is only written if the user allocates a SYSMDUMP data set for the TSO/E session. The system creates a file that is named <i>coredump.pid</i> in the user's working directory (where <i>pid</i> is the process ID for the process that is being dumped) and writes the core dump (SYSMDUMP) in hexadecimal format.
MVS data set name	The dump is written to an MVS data set. The data set name must be fully qualified and can be up to 44 characters. It can be specified in uppercase, lowercase, or both; it is folded to uppercase.
z/OS UNIX file name	The dump is written to a z/OS UNIX file. The file name can be up to 1024 characters and must begin with a slash. The slash refers to the root directory, in which the file is created.

22. The \_BPXK\_JOBLOG environment variable can be used to specify that WTO messages are to be written to an open job log file. The following table lists the allowable values:

Value	Description
<i>nn</i>	Job log messages are written to open file descriptor <i>nn</i> .
STDERR	Job log messages are written to the standard error file descriptor, 2.
NONE	Job log messages are not written. NONE is the default.

The file that is used to capture messages can be changed at any time by calling the `oe_env_np` service and specifying `_BPXK_JOBLOG` with a different file descriptor.

Message capturing is turned off if the specified file descriptor is marked for close on a fork or exec.

Message capturing is process-related. All threads under a given process share the job log file. Message capturing may be initiated by any thread under that process.

Multiple processes in a single address space can each have different files active as the JOBLOG file; some or all of them can share file; and some processes can have message capturing active while others do not.

When the file that is used as a job log is shared by several processes (for example, by a parent and child), the file should be opened for append. Failure to do this causes unpredictable results.

Only files that can be represented by file descriptors may be used as job log files; MVS data sets are not supported.

Message capturing is propagated on a fork() or spawn(). If a file descriptor is specified, the physical file must be the same in order for message capturing to continue in the forked or spawned process. If STDERR is specified, the file descriptor may be remapped to a different physical file.

Message capturing can be overridden on exec() or spawn() by specifying the \_BPXK\_JOBLOG environment variable as a parameter on the exec() or spawn().

Message capturing only works in forked (BPXAS) address spaces.

This is not true job log support: messages that typically go to the JESYSMSG data set are captured, but messages that go to JESMSGLG are not.

23. If the BPXK\_SIGDANGER environment variable is set to YES, the process receives a SIGDANGER signal rather than a SIGTERM signal when an OMVS shutdown is initiated. This might be advantageous for an application that uses the SIGTERM signal for other purposes.
24. When the executable file to be run is a REXX exec, the first argument should be the path name of the REXX exec. Subsequent arguments for the exec can follow this. Each argument should be a string terminated by a null character; that is, the last byte should be X'00'. Each argument length should include this last byte.
25. An environment variable, AUTHPGMLIST, has been created to work with this system call. This environment variable specifies a list that identifies the sanctioned directories or authorized program names. If activated, an additional level of security checking will be performed to ensure that the program being instantiated is coming from an authorized directory in the z/OS UNIX file system or is an authorized MVS program name. For details about the sanction list, see [Using sanction lists in z/OS UNIX System Services Planning](#).
26. The \_BPXK\_SUID\_FORK environment variable specifies whether the setuid indicator is propagated to child address spaces created by the fork service. For more information about \_BPXK\_SUID\_FORK, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).
27. If the caller specifies \_BPXK\_DISABLE\_SHLIB=YES, then future loadhfs() and loadhfs\_extended() system calls will ignore the st\_sharelib attribute and load the program into private storage. If the caller specifies NO (the default) then normal system shared library processing takes place. For more information about \_BPXK\_DISABLE\_SHLIB, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).
28. The \_BPX\_CONTAINER\_POD\_ID environment variable can be used to set the pod ID of the new process image. This identifier is used when running z/OS UNIX-based applications in containers using z/OS Container Platform technology. The rules for specifying the pod ID are as follows:
  - Up to 64 actual pod ID characters are allowed.
  - There is no restriction on the character set.
  - The caller must be permitted to the CONTAINERS resource in the UNIXPRIV class or be a superuser.

If the pod ID is greater than 64 characters, the data is ignored. No other validity or syntax checking is done.
29. The \_BPX\_CONTAINER\_ID environment variable can be used to set the container ID of the new process image. This identifier is used when running z/OS UNIX-based applications in containers using z/OS Container Platform technology. The rules for specifying the container ID are as follows:
  - Up to 64 actual container ID characters are allowed.
  - There is no restriction on the character set.
  - The caller must be permitted to the CONTAINERS resource in the UNIXPRIV class or be a superuser. If the container ID is greater than 64 characters, the data is ignored. No other validity or syntax checking will be done.
30. The \_BPX\_CONTAINER\_QUAL environment variable can be used to set the container qualifier of the new process image. This qualifier is used when running z/OS UNIX-based applications in containers using z/OS Container Platform technology. The rules for specifying the container qualifier are as follows:

- Up to 32 actual container qualifier characters are allowed.
- There is no restriction on the character set.
- The caller must be permitted to the CONTAINERS resource in the UNIXPRIV class or be a superuser.
- A null string indicates that the caller wants to clear an existing container qualifier.

If the container qualifier is greater than 32 characters, the data is ignored. No other validity or syntax checking is done.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“sigpending \(BPX1SIP, BPX4SIP\) — Examine pending signals ” on page 843](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)
- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)

**Note:** The exec service is not related to the exec shell command.

## Characteristics and restrictions

1. The exec service dynamically inserts into a job a new job step that has no allocations associated with it, with the exception of the MVS data sets that may be built into the STEPLIB environment for the new process image.
2. If the exec service is invoked from a process that contains one task, one request block (RB), and no linkage stack entries, the process is ended. This action results in a normal return to the operating system. Almost all forked processes run in this manner. In all other cases, the calling task receives a nonretryable EC6 abend with reason code 0000FFFE to cause it to end. All other thread tasks in the address space that are not subtasks of the calling task receive a 422 abend with reason code 00000181.
3. The user exit is given control while the exec service is still in progress. The user exit should not attempt to use any z/OS UNIX service that alters or terminates the current process (that is, the exec, exit, and kill services). If such services are attempted, the results are unpredictable. Signals cannot be delivered while in the user exit, because the exec service is still in progress and signal delivery is inhibited.
4. If you intend to run a program in an APF-authorized environment, the program that is being run by the exec service should have the APF extended attribute turned on. It should also be linked AC=1. DLLs that are loaded by APF-authorized applications should have the APF extended attribute set on and should be linked AC=0.
5. Any shared memory segments attached to the caller are not attached to the newly created process image. Any shared memory segments attached to the caller are detached and the value of shm\_nattch decremented by the number of shared memory segments attached to the caller. If this is the last process attached to a shared memory segment and a shmctl IPC\_RMID has been issued, the segment is removed from the system.
6. For semaphore users, when the process exec is issued, the SemAdj values are inherited by the new process image.
7. Executing a program from z/OS UNIX causes the program environment to become uncontrolled, unless the program is identified as program controlled. (That is, unless the ST\_PROGCTL attribute is ON for the z/OS UNIX program file.) Running a z/OS UNIX program with the ST\_PROGCTL attribute set to OFF prevents future invocations of authorized programs like Program Access to Data Sets (PADS) programs.

These are programs given special authorization by the installation and by the installed security product (such as RACF) to read or write to protected data sets. In addition, PADS programs should not attempt to load programs from z/OS UNIX with the ST\_PROGCTL attribute OFF, because these programs are considered uncontrolled and could have been modified by users that do not have the same level of authorization as the PADS program.

8. The \_BPXK\_TIMEOUT environment variable is processed for this callable service. It allows applications to specify whether a specific process waiting for terminal input should be timed out. For more information about the \_BPXK\_TIMEOUT environment variable, see [Commonly used environment variables](#) in *z/OS UNIX System Services Planning*.

### Examples

See “BPX1EXC (exec) example” on page 1234.

### MVS-related information

If the exec service is invoked from an address space containing a single process, it tears down the existing process image by ending the currently running job step. It then inserts a new step for the specified file to run in. Any MVS task-related resources that existed in the old job step are cleaned up. The new job step that is created has no allocations associated with it, except for the MVS data sets that may be built into the STEPLIB environment for the new process image. When the newly created job step ends, the flow of the job continues, as it normally does, to the next sequential step in the job, depending on the completion code of the ending step.

If the exec service is invoked from an address space containing multiple processes, the following characteristics apply:

- If the calling process does not have any subtasks that are part of another process, and if the calling process was created via a call to the attach\_execmvs or attach\_exec service, only the initial thread task of the process and all of its subtasks are terminated, and a new task is attached to the parent process creator task to run the specified program. The initial thread task in such a process is the task that was created as a result of the call to the attach\_execmvs or attach\_exec service. This call to the exec service does not result in the ending of any other tasks in the calling job step, nor does it end other processes in the same address space.
- If the calling process has any subtasks that are part of another process, or if the calling process was not created via a call to the attach\_exec or attach\_execmvs service, the exec invocation is not allowed to prevent the unexpected termination of other processes in the address space. The caller receives a return code and reason code detailing the error.

If the exec service is invoked after a successful setuid that changes the MVS identity and the \_BPX\_JOBNAME environment variable was not specified, the job name of the new process image is set to the user ID associated with the new UID specified on the setuid invocation.

## execmvs (BPX1EXM, BPX4EXM) — Run an MVS program

---

### Function

The execmvs service runs an MVS executable program that is in the link pack area (LPA) or LNKLST concatenation. If it is invoked from an address space that contains multiple processes, the program can come from a STEPLIB.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task

Operation	Environment
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1EXM):</b>	31-bit
<b>AMODE (BPX4EXM):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1EXM, (Program_name_length,
               Program_name,
               Argument_length,
               Argument,
               Exit_routine_address,
               Exit_parameter_list_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4EXM with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Program\_name\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the name of the MVS program.

### Program\_name

Supplied parameter

#### Type:

Character string

#### Character set:

No restriction

#### Length:

Specified by the Program\_name\_length parameter

The name of a field that contains the name of the MVS program that is to be run. The MVS program name must conform to the naming conventions for members of MVS partitioned data sets (PDSs). The program name is 1 - 8 characters long; the program name is the member name without any qualifiers.

The specified Program\_name must be in uppercase.

### Argument\_length

Supplied parameter

#### Type:

Integer

**Length:**

Fullword

The argument can be from 0 to 4096 bytes long except for unauthorized callers that are calling authorized programs. For unauthorized callers that are calling authorized programs, the argument can be from 0 to 100 bytes long. To allow an unauthorized caller to pass an argument greater than 100 bytes to a program, define a BPX.EXECMVSAPF.*program\_name* FACILITY class profile for that program

**Note:** If an unauthorized caller attempts to pass an argument greater than 100 bytes to an authorized program, the caller will abend with the ABENDEC6 reason code 0B26C048. To allow the caller to execute the program with an argument greater than 100 bytes, define a BPX.EXECMVSAPF.*program\_name* profile.

**Argument**

Supplied parameter

**Type:**

Character string

**Length:**

Specified by the Argument\_length parameter

The name of a field of length Argument\_length that contains the argument that is to be passed to the MVS program.

The data that is contained in the Argument parameter should not include pointers to private storage. The execmvs service frees all private storage while cleaning up the previous job step.

**Exit\_routine\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be invoked, define Exit\_routine\_address as the name of a fullword (doubleword) that contains 0.

**Exit\_parameter\_list\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list.

The value that is contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be invoked or does not require parameters, define Exit\_parameter\_list\_address as the name of a fullword (doubleword) that contains 0. Currently the exit must be RMODE 31, and therefore the address must reside below the 2-gigabyte bar.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the execmvs service returns -1 if it is not successful. If it is successful, the execmvs service does not return.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the execmvs service stores the return code. The execmvs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The execmvs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>E2BIG</b>	The number of bytes used by the new process image's argument list is greater than the system-imposed limit of 4096 bytes. The following reason code can accompany the return code: JRMVSAArgTooBig.
<b>EFAULT</b>	The user exit program checked. The following reason code can accompany the return code: JRExitRtnError.
<b>ENAMETOOLONG</b>	The specified MVS program name is too long. The length that is specified by Program_name_length is longer than 8 bytes.
<b>ENOENT</b>	The specified MVS program was not found in the link pack area or in a link list data set, LNKLST; or the program name argument points to an empty string. STEPLIB needs to be included in a multiprocess environment. The following reason code can accompany the return code: JRExecNmLenZero.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the execmvs service stores the reason code. The execmvs service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes for execmvs**

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes execmvs to run a program, the new process image that is created to run the program inherits the privilege of the set-user-ID or set-group-ID program.
2. The following characteristics of the calling process are changed when the new executable is given control by the execmvs service:
  - The prior process image is replaced with a new process image for the executable program that is to be run.
  - All open files that are marked close-on-exec and all open directory streams are closed.
  - All signals that have sigaction settings are reset to their default actions.
3. The input that is passed to the MVS executable file by the service is consistent with the input that is passed to MVS programs. On input, the MVS program receives a single-entry parameter list that is pointed to by register 1. The high-order bit of the sole parameter entry is set to 1.

The sole parameter entry is the address of a 2-byte length field followed by an argument string. The length field describes the length of the data that follows it. If a null argument and argument length are specified in the call, the length field specifies 0 bytes on input to the executable file.



4. The call can invoke both unauthorized and authorized MVS programs:
  - Unauthorized programs receive control in problem program state, with PSW key 8.
  - Authorized programs receive control in problem program state, with PSW key 8 and APF authorization.
5. The register usage on entry to the user exit in AMODE 31 is as follows:
  - R0: Undefined.
  - R1: Address of the user exit parameter list, as specified by the caller of the execmvs service.
  - R2–R12: Undefined
  - R13: Address of a 96-byte work area in the same key as the caller of the execmvs service.
  - R14: The return address from the user exit to the execmvs service. This address must be preserved by the user exit.
  - R15: Address of the user exit.
6. The register usage on entry to the user exit in AMODE 64 is as follows:
  - R0: Undefined.
  - R1: 64-bit address of the user exit parameter list, as specified by the caller of the execmvs service.
  - R2–R12: Undefined
  - R13: Address of a 96-byte work area in the same key as the caller of the execmvs service.
  - R14: The return address from the user exit to the execmvs service. This address must be preserved by the user exit.
  - R15: R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.
7. When the exec or execmvs service is called in any environment except single task, single RB, and no linkage stack, z/OS UNIX issues a quiesce\_force to terminate all of its subtasks. The subtasks receive a 422 abend with a reason code of 000001A0.
8. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the execmvs service are propagated to the new process image if the data sets that they represent are found to be cataloged. Uncataloged data sets are not propagated to the new process image. This causes the program that is invoked to run with the same MVS program search order as its invoker.
9. To support the creation and propagation of a STEPLIB environment to the new process image, the execmvs service allows for the specification of a STEPLIB environment variable. If the STEPLIB environment variable is not specified, the default behavior of the execmvs service is the same as if STEPLIB=CURRENT were specified.
 

If the program that is to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets that are to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For detailed information about the sanction list, see [Using sanction lists in z/OS UNIX System Services Planning](#). For information about STEPLIB performance considerations, see [Controlling use of STEPLIBs in z/OS UNIX System Services Planning](#).
10. If the calling task is in a WLM enclave, the new process image task is joined to the same WLM enclave. This allows WLM to manage the old and new process images as one business unit of work entity for system accounting and management purposes.
11. If an unauthorized caller attempts to pass an argument greater than 100 bytes to an authorized program, the caller will abend with the ABENDEC6 reason code 0B26C048 or ABEND306 reason code 44. If the caller needs to be allowed to execute the program with an argument greater than 100

bytes, then a BPX.EXECMVSAF.*program\_name* profile should be defined or have the program owner rebind the program specifying the LONGPARM option.

12. If the caller specifies `_BPXK_DISABLE_SHLIB=YES`, then future `loadhfs()` and `loadhfs_extended()` system calls will ignore the `st_sharelib` attribute and load the program into private storage. If the caller specifies NO (the default) then normal system shared library processing takes place. For more information about `_BPXK_DISABLE_SHLIB`, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).

### **Related services**

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)

### **Characteristics and restrictions**

- When the `execmvs` service is called from any process except one that was created by the `attach_exec` or `attach_execmvs` service, the program must be located either in the link pack area (LPA) or in a link list data set (LNKLST).
- When the `execmvs` service is called from a process that was created via the `attach_exec` or `attach_execmvs` service, the specified program can be located in the link pack area, in a link list data set, job library, step library, or task library. The program search order that is followed is identical to that of the MVS Attach service when the EP parameter is specified.
- If the `execmvs` service is invoked from a process that contains one task, one request block (RB), and no linkage stack entries, the process is ended by an SVC 3 instruction. This action results in a normal return to the operating system. Almost all forked processes run in this manner. In all other cases, the system ends all tasks (threads) in the caller with a nonretryable 422 abend, reason code 000001A0.
- The user exit cannot invoke any z/OS UNIX services. If it attempts to invoke a z/OS UNIX service, the service fails or the caller is abended, depending on the service that is attempted. Signals cannot be delivered to the caller of the `exec` service while the user exit is in control.
- The program that is invoked by the `execmvs` service must be enabled to run in 31-bit addressing mode (AMODE=31).

### **Examples**

See [“BPX1EXM \(execmvs\) example” on page 1235](#).

### **MVS-related information**

Because the service must create a new process image for the specified program to run within, the prior process image is completely cleaned up. In MVS terms, the system ends a step within a job and then inserts a new step for the specified program to run in. Any MVS task-related resources that existed in the old job step are cleaned up. The new job step that is created has no allocations that are associated with it, except for the MVS data sets that might be built into the STEPLIB environment for the new process image. When the newly created job step ends, the flow of the job continues, as it normally does, to the next sequential step in the job, depending on the completion code of the ending step.

## **\_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup**

### **Function**

The `_exit` callable service ends the calling thread task and all its subtasks. In most environments, this results in the ending of the process, with the specified status being reported to its parent.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1EXI):</b>	31-bit
<b>AMODE (BPX4EXI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1EXI, (Status_field)
```

AMODE 64 callers use BPX4EXI with the same parameter.

## Parameters

### Status\_field

Supplied parameter

#### Type:

Structure

#### Length:

4 bytes

The name of a 4-byte status field. If the call to `_exit` results in a process ending and contents of the status field conform to the allowable exit status values, the service provides the contents to the parent when a wait service is issued. For a mapping of the status field and a description of the conforming status values, see [“BPXYWAST — Map the wait status word”](#) on page 1172.

## Usage notes

1. A call to `_exit` results in the ending of the calling task and all its subtasks, and the cleaning up of their associated MVS and z/OS UNIX resources. In most environments, this results in the ending of the calling process.
2. In some environments the call to `_exit` does not result in a process ending. An example of such an environment is the TSO/E TMP environment, where multiple MVS tasks can be concurrently dubbed as threads. A call to the `_exit` service from one of these threads results only in the ending of the calling thread task and its subtasks. In such an environment, if only one task is currently dubbed as a thread, a call to the `_exit` service from this thread task ends the process.
3. The ending of a process results in the following actions:
  - All file descriptors and directory streams that are open in the ending process are closed. Open file descriptors are inherited by the child. Literally speaking, the child did not open the file, yet it will still be closed.

- If the parent of the ending process has issued a wait call and is waiting for the ending process to end, has not used sigaction to set its SA\_NOCLDWAIT flag for the SIGCHLD signal, and has not set the action for SIGCHLD to ignore, the status is returned to the parent at once.

If the parent of the ending process is not waiting, has not used sigaction to set its SA\_NOCLDWAIT flag for the SIGCHLD signal, and has not set the action for SIGCHLD to ignore, the status is saved. It is returned to the parent if the parent later issues a wait call for the now-ended child.

- If the parent of the ending process has set the SA\_NOCLDWAIT flag for the SIGCHLD signal, or has set the action for SIGCHLD to ignore, the status is discarded and will not be seen by the parent if the parent issues a wait. The ending process is assigned the parent process ID of the initialization process (whose process ID is 1) that frees the PID and system resources that are associated with the ending process.

If the parent of the ending process does not later wait for the ending process, and has not used sigaction to set its SA\_NOCLDWAIT flag for the SIGCHLD signal, and has not set the action for SIGCHLD to ignore, the ending process's ID (PID) remains in use until the parent ends. Because the number of process IDs is a limited system resource, user and system availability for process IDs may be affected.

- If the ending process is a session leader, the controlling terminal is disassociated from the session. The controlling terminal can then be acquired by a new controlling process.
  - Child processes of a process that ends are assigned the parent process ID of the initialization process (whose process ID is 1). The status of these child processes is reported to the initialization process that frees the PID and system resources associated with the ending process.
  - A SIGCHLD signal is sent to the parent of the ending process.
  - Ending a process does not end its child processes directly, however; under the following circumstances a SIGHUP signal is sent to a child process that can cause a child process to end:
    - If the ending process is a controlling process, a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the caller.
    - If the ending process is a dubbed process that has not been a controlling process of a terminal session; for example, a batch job step that has issued z/OS UNIX service calls; a SIGHUP signal is sent to each process in the ending process's process group.
    - If ending a process leaves a process group orphaned and any member of that process group is stopped, each member of the process group is sent a SIGHUP signal followed by a SIGCONT signal.
4. If the ending of the calling task results in the ending of a job step, the specified status code is used as the completion code for the ending job step.
  5. The \_exit service does not return to the caller. If it cannot complete its processing successfully, the caller receives an EC6 abend.
  6. If the caller specifies an incorrect exit status value, the caller receives an EC6 abend with an appropriate reason code identifying the error.
  7. If you are going to use this service in a multiple-pthread environment, see [Appendix H, “Using threads with callable services,”](#) on page 1421.
  8. Each shared-memory segment attached to the calling process is detached, and the value of the number of processes attached to each detached segment (shm\_nattch) is decremented by 1. If this is the last process attached to a shared memory segment and shmctl IPC\_RMID has been issued for the shared memory segment, the segment is removed from the system.
  9. When the process is terminated, the semadj values are applied to the semaphores. Adjustments to each semaphore set are made atomically.

## Related services

- [“close \(BPX1CLO, BPX4CLO\) — Close a file ”](#) on page 121
- [“mvspocclp \(BPX1MPC, BPX4MPC\) — Clean up kernel resources ”](#) on page 484

- “wait (BPX1WAT, BPX4WAT) — Wait for a child process to end ” on page 984

**Note:** The \_exit service is not related to the exit shell command and is different from the exit() ANSI C routine.

## Characteristics and restrictions

If the \_exit service is invoked with a normal exit status completion code from a task that has no subtasks, one request block (RB), and no linkage stack entries, the task ends with an SVC 3 instruction. This action results in a normal return to the operating system. Almost all forked processes end in this manner. In all other cases, the calling task receives a nonretryable EC6 abend with a reason code that varies with the type of exit status specified. If the exit status value indicates that the process is to end with:

- A normal exit status code, an abend reason code of 0000FFFF is received.
- An ending signal, an abend reason code of 0000FFxx is received, where xx is the signal number specified in the exit status.
- A terminating signal with a core dump to be taken, an abend reason code of 0000FDxx is received, where xx is the signal number specified in the exit status.

All subtasks of the calling thread task receive a 33E abend when the calling thread task is abended.

If the calling thread task was created with the pthread\_create service, the initial pthread-creating task abends with a 422 abend code, and reason code 000001xx. The value of xx is the signal number if signal exit status is specified, or 82 if a normal exit status is specified.

For a detailed description of the conforming exit status values see “BPXYWAST — Map the wait status word” on page 1172.

## Examples

See “BPX1EXI (\_exit) example” on page 1235.

# extlink\_np (BPX1EXT, BPX4EXT) — Create an external symbolic link

---

## Function

The extlink\_np service creates a symbolic link to an external name. A file named Link\_name, of type "symbolic link" is created. The content of the symbolic link file is the external name specified in Ext\_name.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1EXT):</b>	31-bit
<b>AMODE (BPX4EXT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1EXT, (Ext_name_length,  
              Ext_name,  
              Link_name_length,  
              Link_name,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4EXT with the same parameters.

## Parameters

### **Ext\_name\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword containing the length of Ext\_name. The Ext\_name can be up to 1023 bytes long.

### **Ext\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Ext\_name\_length parameter

The name of a field containing the external name for which you are creating a symbolic link. An external name is the name of an object outside the hierarchical file system.

### **Link\_name\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword containing the length of Link\_name. The Link\_name can be up to 1023 bytes long; each component of the name (between delimiters) can be up to 255 bytes long.

### **Link\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by Link\_name\_length parameter

The name of a field containing the symbolic link being created.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the extlink\_np service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the extlink\_np service stores the return code. The extlink\_np service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The extlink\_np service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search a directory in the Link_name, or does not have permission to write in the directory to contain the symbolic link file.
<b>EEXIST</b>	Link_name already exists.
<b>EINVAL</b>	Parameter error. Possible reasons are: <ul style="list-style-type: none"> <li>Ext_name_length exceeds the maximum allowed.</li> <li>Ext_name_length is zero.</li> <li>Link_name has a slash as its last component, which indicates that the preceding component is a directory. A symbolic link cannot be a directory.</li> </ul> <p>The following reason codes can accompany the return code: JRInvalidSymLinkLen, JREndingSlashSymLink.</p>
<b>ELOOP</b>	A loop exists in symbolic links that are encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
<b>ENAMETOOLONG</b>	Link_name is longer than 1023 characters, or some component of that name is longer than 255 characters. Name truncation is not supported.
<b>ENOSPC</b>	The directory in which the entry for the symbolic link is being placed cannot be extended; not enough space remains in the file system.
<b>ENOTDIR</b>	A component of the path prefix of Link_name is not a directory.
<b>EROFS</b>	The requested operation requires writing in a directory on a read-only file system.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the extlink\_np service stores the reason code. The extlink\_np service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The extlink\_np service creates an external symbolic link (Link\_name) with the object you specify by Ext\_name.
2. The object that is identified by Ext\_name need not exist when the symbolic link is created, and refers to an object outside a hierarchical file system.
3. The external name in an external symbolic link is not resolved. The Link\_name that is contained cannot be used as a directory component of a path name.

**Related services**

- [lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name](#) on page 409
- [readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link](#) on page 666
- [symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name](#) on page 900

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1EXT \(extlink\\_np\) example”](#) on page 1235.

## **faccessat (BPX1FAA, BPX4FAA) — Checks user's permissions of a file relative to a directory file descriptor**

---

**Function**

The faccessat callable service operates the same way as the access callable service except for the following differences:

- If the path name given in Pathname is relative, then it is interpreted relative to the directory referred to by the file descriptor dirfd (rather than relative to the current working directory of the calling process, as is done by access() for a relative path name).
- If Pathname is relative and dirfd is the special value AT\_FDCWD, then it is interpreted relative to the current working directory of the calling process, like access(). If Pathname is absolute, then dirfd is ignored.

**Requirements**

<b>Operation</b>	<b>Authorization</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1FAA):</b>	31-bit



Operation	Authorization
<b>AMODE (BPX4FAA):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax has the following format:

```
CALL BPX1FAA, (Pathname_length,
               Pathname,
               Accessat_mode,
               Dirfd
               Flags
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FAA with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the path name of the file whose mode you want to change.

### Pathname

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. The length of this field is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory. When Dirfd is AT\_FDCWD, the working directory is the current working directory. Otherwise, Dirfd contains the file descriptor for the starting directory.

### Access\_mode

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword field that indicates the access mode to be tested. This field is mapped by the BPXYACC macro. The values for this field are:

**ACC\_F\_OK**

Test for file existence. This is the default value.

**ACC\_R\_OK**

Test for permission to read.

**ACC\_W\_OK**

Test for permission to write.

**ACC\_X\_OK**

Test for permission to execute or search.

**ACCWAIT**

If an asynchronous mount is in progress, wait for it to complete.

**ACCDEVNO**

Return the devno of the file in Return\_value.

**ACCEFFID**

Use the effective ID rather than the real ID to check for permission.

**Dirfd**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains a file descriptor for the starting directory to be used for the make node request. The special value AT\_FDCWD that was specified with a relative path name, indicates to use the current working directory to interpret the path. Otherwise, a relative pathname will be interpreted relative to the directory referred to by this file descriptor. If path name is absolute, this parameter is ignored.

**Flags**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword field to further define the faccessat. Flags can either be 0 or include the following, which are defined in the BPXYFCTL macro.

**AT\_SYMLINK\_NOFOLLOW**

When specified with a pathname of a symbolic link, the permission change is performed on the link itself, as opposed to the default behavior where the link is dereferenced.

**AT\_EACCESS**

Perform access checks using the effective user and group IDs. By default, faccessat() uses the real IDs

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the faccessat service returns 0 if the request completes successfully (that is, the file exists or faccessat is permitted), or -1 if the request is not successful, or the file cannot be accessed in the specified way.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the faccessat service stores the return code. The faccessat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The faccessat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have appropriate permissions to access the file in the ways specified by the Access_Mode parameter, or does not have search permission for some component of the Pathname prefix.
<b>EBADF</b>	Dirfd is not a valid descriptor for AT_FDCWD.
<b>EINVAL</b>	The Access_Mode parameter is incorrect. The following reason code unique to the faccessat service can accompany the return code: JRInvalidAMODE.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or some component of the pathname is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file named Pathname was found, or no Pathname was specified. The following reason code unique to the faccessat service can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Not a directory returned for the following: <ul style="list-style-type: none"> <li>• Some component of Pathname is not a directory. (JRCompNotDir)</li> <li>• The Dirfd parameter does not refer to a directory. (JRAtNotDirFD)</li> </ul>
<b>EROFS</b>	The Access_Mode parameter is testing for write access to a read-only file system. The following reason code unique to the faccessat service can accompany the return code: JRReadOnlyFS.

Refer to [“access \(BPX1ACC, BPX4ACC\) — Determine if a file can be accessed”](#) on page 24.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the faccessat service stores the reason code. The faccessat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. Testing for file permission is based on the real user ID (UID) and real group ID (GID), unless the ACCEFFID bit has been set on. In that case, the effective ID is used for the test.
2. The caller can test for the existence of a file, or for faccessat to the file, but not for both.
3. In testing for permission, the caller can test for any combination of read, write, and execute permission. If the caller is testing a combination of permissions, Return\_value indicates failure if any one of the accesses is not permitted.
4. If the caller has appropriate privileges (see [“Authorization” on page 8](#)), the faccessat test is successful even if the permission bits are off, except when testing for execute permission. When the caller tests for execute permission, at least one of the execute permission bits must be on for the test to be successful.
5. If the faccessat\_mode parameter is zero, the service performs the existence test, ACC\_F\_OK.

## Related services

- [“access \(BPX1ACC, BPX4ACC\) — Determine if a file can be accessed” on page 24](#)
- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1FAA \(faccessat\) example” on page 1235](#) and [“BPX4FAA \(faccessat\) example” on page 1329](#).

# fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor

---

## Function

The fchattr service modifies the attributes that are associated with a file. It can be used to change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, file size, and file tag. It can also be used to set the initial security label for a file or directory. You identify the file by its file descriptor.

For the corresponding service that uses a path name, see [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory ” on page 82](#).

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN

<b>Operation</b>	<b>Environment</b>
<b>AMODE (BPX1FCR)</b>	31-bit
<b>AMODE (BPX4FCR)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1FCR, (File_descriptor,
               Attributes_length,
               Attributes,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FCR with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the file descriptor of the file whose attributes that you want to change.

### Attributes\_length

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the length of the area that contains the attributes that you want to change.

### Attributes

Supplied parameter.

#### Type

Structure.

#### Length

Specified by the Attributes\_length parameter.

The name of the area that contains the attributes that you want to change. The area is mapped by BPXYATT. For information about the content of this area, see [“BPXYATT — Map file attributes for chattn and fchattn” on page 1057](#).

For extended attributes, the area is mapped by the attx structure that is located in the same BPXYATT mapping. Storage for the attribute value must be contiguous with the attx structure

and the `attributes_length` argument is the length of the `attx` structure plus the length of the attribute value.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword where the `fchattr` service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the `fchattr` service stores the return code. The `fchattr` service returns the return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The `fchattr` service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The calling process did not have appropriate permissions. Possible reasons include: <ul style="list-style-type: none"> <li>• The calling process was attempting to set access time or modification time to current time; the effective UID of the calling process does not match the owner of the file; the process does not have write permission for the file; or the process does not have appropriate privileges. (For the appropriate privileges, see <a href="#">“Authorization”</a> on page 8.).</li> <li>• The calling process was attempting to change the file size; the calling process does not have write permission for the file.</li> </ul>
<b>EBADF</b>	The <code>File_descriptor</code> parameter is not a valid file descriptor.
<b>EFBIG</b>	Attempting to change the size of a file, the specified length is greater than the maximum file size limit for the process. The following reason code can accompany the return code: <code>JRWriteBeyondLimit</code> .
<b>EEXIST</b>	A request to create an extended attribute was specified, but the attribute is already set. The following reason code can accompany the return code: <code>JrAttxExists</code> .
<b>EINVAL</b>	The extended attribute operation is invalid due to the values that are being passed. The input length for the <code>ATTX</code> parameter might be insufficient or the subcommand <code>AttxSubCmd</code> is invalid. The following reason codes can accompany the return code: <code>JRInvMinArgLenAttx</code> , <code>JRInvArgLenAttx</code> , <code>JRInvAttxCmd</code> .
<b>EMVSERR</b>	An MVS environmental error was detected. The following reason code can accompany the return code: <code>JrSeclabelClassInactive</code> .
<b>ENOATTR</b>	For the replace or remove function, the specified extended attribute cannot be set. <code>ENOATTR</code> is also set if the passed in attribute name, <code>AttxName</code> , is empty. The following reason codes can accompany the return code: <code>JrAttxName</code> and <code>JrAttxUnavail</code> .

Return_code	Explanation
<b>ENOSYS</b>	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.
<b>ENOTSUP</b>	The request against an extended attribute is not supported. The following reason codes can accompany the return code: JRInvAttxFType, JrSetAttxTrustClas, JRAttxNoSup, and JRSetAttxNameSpc.
<b>EPERM</b>	<p>The operation is not permitted for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• The calling process was attempting to change the mode or the file format; the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see “Authorization” on page 8).</li> <li>• The calling process was attempting to change the owner, and the calling process does not have appropriate privileges.</li> <li>• The calling process was attempting to change the general attribute bits, and the calling process does not have write permission for the file.</li> <li>• The calling process was attempting to set a time value (not current time); the effective user ID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges.</li> <li>• The calling process was attempting to set the change time or reference time to current time, and the calling process does not have write permission for the file.</li> <li>• The calling process was attempting to change auditing flags; the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges.</li> <li>• The calling process was attempting to change the security auditor's auditing flags, and the user does not have auditor authority.</li> <li>• Attributes indicate that the security label is to be set, and either the calling process does not have RACF SPECIAL authorization and appropriate privileges or there is already a security label that is associated with the file.</li> </ul>
<b>ERANGE</b>	The length of the extended attribute value to be set, AttxValLn, does not match the attribute size itself. For example, ERANGE can occur when a 4-byte attribute is being set to a 2-byte or 6-byte value. The following reason code can accompany the return code: JrInvAttxValLen.
<b>EROFS</b>	The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.
<b>Reason_code</b>	Returned parameter.
<b>Type</b>	Integer
<b>Length</b>	Fullword
<p>The name of a fullword where the fchattr service stores the reason code. The fchattr service returns a reason code only if the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see <a href="#">Reason codes</a> in <i>z/OS UNIX System Services Messages and Codes</i>.</p>	

## Usage notes

Table 4. Attribute fields that can be modified by fchattr		
Set flags	Attribute fields input	Description
<b>ATTMODECHG</b>	ATTMODE	Set the mode according to the value in ATTMODE. See <a href="#">“fchmod (BPX1FCM, BPX4FCM) – Change the mode of a file or directory by descriptor”</a> on page 204.
<b>ATTOWNERCHG</b>	ATTUID ATTGID	Set the owner user identifier (UID) and group identifier (GID) to the values specified in ATTUID and ATTGID. See <a href="#">“chown (BPX1CHO, BPX4CHO) – Change the owner or group of a file or directory”</a> on page 104.
<b>ATTSETGEN</b>	ATTGENVALUE ATTGENMASK	Only the bits corresponding to the bits set ON in the ATTGENMASK are set to the value (ON or OFF) in ATTGENVALUE. Other bits are unchanged.
<b>ATTTRUNC</b>	ATTSIZE	Change the file size to ATTSIZE bytes. See <a href="#">“ftruncate (BPX1FTR, BPX4FTR) – Change the size of a file”</a> on page 249.
<b>ATTATIMECHG</b>	ATTATIME	If ATTLTP64TIMES is not set, set the access time of the file to the value specified in ATTATIME. If ATTLTP64TIMES is set, set the access time of the file to the value specified in ATTATIME64, which is a doubleword field.
<b>ATTATIMETOD</b>	None	Set the access time of the file to the current time.
<b>ATTMTIMECHG</b>	ATTMTIME	If ATTLTP64TIMES is not set, set the modification time of the file to the value specified in ATTMTIME. If ATTLTP64TIMES is set, set the modification time of the file to the value specified in ATTMTIME64, which is a doubleword field.
<b>ATTMTIMETOD</b>	None	Set the Modification time of the file to the current time.
<b>ATTMAAUDIT</b>	ATTAUDITORAUDIT	Set the security auditor's auditing flags to the value specified in ATTAUDITORAUDIT. See <a href="#">“fchaudit (BPX1FCA, BPX4FCA) – Change audit flags for a file by descriptor”</a> on page 199.
<b>ATTMUAUDIT</b>	ATTUSERAUDIT	Set the User's auditing flags to the value specified in ATTUSERAUDIT. See <a href="#">“fchaudit (BPX1FCA, BPX4FCA) – Change audit flags for a file by descriptor”</a> on page 199.



Table 4. Attribute fields that can be modified by fchattr (continued)

Set flags	Attribute fields input	Description
<b>ATTCTIMECHG</b>	ATTCTIME	If ATTLP64TIMES is not set, set the change time of the file to the value specified in ATTCTIME. If ATTLP64TIMES is set, set the change time of the file to the value specified in ATTCTIME64, which is a doubleword field.
<b>ATTCTIMETOD</b>	None	Set the Change Time of the file to the current time.
<b>ATTREFTIMECHG</b>	ATTREFTIME	If ATTLP64TIMES is not set, set the reference time of the file to the value specified in ATTREFTIME. If ATTLP64TIMES is set, set the reference time of the file to the value specified in ATTREFTIME64, which is a doubleword field.
<b>ATTREFTIMETOD</b>	None	Set the Reference Time of the file to the current time.
<b>ATTFILEFMTCHG</b>	ATTFILEFMT	Set the File Format of the file to the value specified in ATTFILEFMT.
<b>ATTCHARSETIDCHG</b>	ATTFILETAG	Set the file tag. See BPXYSTAT ( <a href="#">“BPXYSTAT — Map the response structure for stat” on page 1162</a> ) for file tag mapping.
<b>ATTSECLABELCHG</b>	ATTSECLABEL	Set the initial security label for a file or directory.

1. Flags in the Attributes parameter are set to indicate which attributes should be updated. To set an attribute, turn the corresponding Set Flag on, and set the corresponding Attributes Field according to [Table 4 on page 194](#). Multiple attributes can be changed at the same time.

Clear the **Set Flag** field before any bits are turned on. It is an error if any of the reserved bits in the flag field are turned on.

2. Some of the attributes that are changed by the fchattr service can also be changed by other services. See the related service (listed in [Table 4 on page 194](#)) for a detailed description.
3. Changing mode (ATTMODECHG = ON):
  - The file mode field in Attributes is mapped by the BPXYMODE macro. For more information about BPXYMODE, see [“BPXYMODE — Map the mode constants” on page 1106](#). For more information about the values for file type, see [“BPXYFTYP — File type definitions” on page 1078](#).
  - File descriptors that are open when the fchattr service is called retain the access permission they had when the file was opened.
  - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
  - Setting the set-group-ID-on-execution permission (in mode) means that when this file is run through the exec service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

The set-group-ID-on-execution permission is set to zero if both the caller does not have appropriate privileges and the GID of the file's owner does not match the effective GID or one of the supplementary GIDs of the caller.

- Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

#### 4. Changing owner (ATTOWNERCHG = ON):

- To change the owner UID of a file, the caller must have appropriate privileges.
- To change the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:
  - The effective UID of the caller matches the file's owner UID.
  - The Owner\_UID value that is specified in the change request matches the file's owner UID.
  - The Group\_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
- When owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
- When the owner is changed, both UID and GID must be specified as they are to be set, or set to -1 if the value is to remain unchanged. If only one of these values is to be changed, the other one can be set to its present value or to -1 to remain unchanged.

#### 5. Changing general attribute bits (ATTSETGEN = ON):

- For General Attribute bits to be changed, the calling process must have write permission for the file.

#### 6. Changing the size of a file (ATTTRUNC = ON):

- The resizing of a file to ATTSIZE bytes changes the file size to ATTSIZE, beginning from the first byte of the file. If the file was previously larger than ATTSIZE bytes, the data from ATTSIZE to the original end of file is removed. If the file was previously shorter than ATTSIZE, bytes between the old and new lengths are read as zeros.

Full blocks are returned to the file system so that they can be used again.

The file offset is not changed.

- When a file is changed successfully, it clears the set-user-ID, the set-group-ID, and the save-text (sticky bit) attributes of the file unless the caller has appropriate privileges.
- The changing of a file to ATTSIZE bytes, where ATTSIZE is greater than the soft file size limit for the process, will fail with EFBIG. The SIGXFSZ signal is also generated for the process.
- If write access is removed at some time after the File\_descriptor was opened for writing, a change request will fail with EACCES. In such a case, a call to [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file” on page 249](#) could be used to change the file size.

#### 7. Changing times:

- All time fields in Attributes are in POSIX format.
- For the Access Time or the Modification Time to be set explicitly (ATTATIMECHG = ON or ATTMTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the Access Time or Modification Time to be set to the current time (ATTATIMETOD = ON or ATTMTIMETOD = ON), the effective ID must match the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.
- For the Change Time or the Reference Time to be set explicitly (ATTCTIMECHG = ON or ATTREFTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the Change Time or Reference Time to be set to the current time (ATTCTIMETOD = ON or ATTREFTIMETOD = ON), the calling process must have write permission for the file.

- For any time field (**atime**, **mtime**, **ctime**, **reftime**), if both current time and specific time are requested (for example, ATTCTIMETOD = ON and ATTCTIMECHG = ON), the current time is set.
  - When any attribute field is changed successfully, the file's change time is updated as well.
8. Changing auditor audit flags (ATTMAAUDIT = ON):
- For auditor audit flags to be changed, the user must have auditor authority. The user with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.
- Auditor authority is established by issuing the TSO/E command ALTUSER Auditor.
9. Changing user audit flags (ATTMUAUDIT = ON):
- For the user audit flags to be changed, the user must have appropriate privileges (see [“Authorization” on page 8](#)) or be the owner of the file.
10. Changing file format (ATTFILEFMTCHG = ON):
- The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
  - The attribute that is specified in ATTFILEFMT is the same attribute that is set by the FILEDATA=TEXT parameter on a DD statement.
11. Changing the file tag (ATTCHARSETIDCHG=ON):
- A file tag can be set for regular, FIFO, and character special files. If the DeferTag bit is on in the file tag, the file must be empty.
12. Changing the security label (ATTSECLABELCHG=ON):
- For the security label to be changed, the user must have RACF SPECIAL authorization and appropriate privileges (see [“Authorization” on page 8](#)), and no security label must exist on the file. Only an initial security label can be set. An existing security label cannot be changed. The function will successfully set the security label if the RACF SECLABEL class is active. If the SECLABEL class is not active, a return code of EMVSERR will be returned.
13. Changing extended attributes:
- To add, remove, or modify extended attributes, use the Extended Attributes control block ATTX in place of the standard ATT control block. For mapping to ATTX, see [“BPXYATT — Map file attributes for chattr and fchattr” on page 1057](#). Rather than turning on flags within the control block like ATT, ATTX supports an attribute name field, AttxName, to specify which attribute to take action against. The ATTX mapping is variable length with the attribute value, AttxVal, field being a CHAR(\*) value. AttxValLn field should be set to the length of the attribute value.
  - The subcommand field, AttxSubcmd, controls whether this service call will SET or REMOVE the attribute. If GET, LIST, or any other value is specified then the call will fail with JRInvAttrCmd.
  - The AttxFlags AttxfReplace and AttxfCreate are used to validate the request against the extended attribute. AttxfReplace checks whether the attribute is currently set and able to be replaced. If not set, this service call returns reason code JrAttrUnavail. AttxfCreate will check whether the attribute is not set and able to be created (set). If already set, BPX1CHR/BPX4CHR will return reason code JrAttrExists.
  - The AttxfLset flag controls whether the operation affects a symlink's attributes or the attributes of the symlink's target. When this flag is ON, the operation affects the symlink. When the flag is OFF, the operation affects the symlink's target.
  - Changing the general attributes is not supported by fchattr (BPX1FCR,BPX4FCR) and lchattr (BPX1LCR, BPX4LCR). General attributes include ATTNODELFILES, ATTSHARELIB, ATTNOSHAREAS, ATTAPFAUTH, and ATTPROGCTL, with corresponding extended attributes trusted.apfauth, trusted.sharelib, trusted.progctl, and system.noshareas.
  - The SET/REMOVE function affects the attribute information that also exists in BPXYSTAT. The following table provides a correspondence between the information in BPXYATT and BPXYSTAT.

Table 5. Correspondence between BPXYATT and BPXYSTAT

Xattr name	Existing SET attribute in BPXYATT	Existing GET attribute in BPXYSTAT	Restrictions and comments
trusted.apfauth	attapfauth	ST_APFAUTH	Set and remove functions that are limited according to “general flags” restrictions.
trusted.sharelib	attsharelib	ST_SHARELIB	Set and remove functions that are limited according to “general flags” restrictions.
trusted progctl	attprogctl	ST_PROGCTL	Set and remove functions that are limited according to “general flags” restrictions.
system.noshareas	attnoshareas	ST_NOSHAREAS	Set and remove functions that are limited according to “general flags” restrictions.
system.filefmt	attfilefmt	ST_FILEFMT	None.
system.filetag	attfiletag	ST_FILETAG	None.
system.seclabel	attseclabel	ST_SECLABEL	None.
system.useraudit	attuseraudit	ST_USERAUDIT	None.
system.auditoraudit	attauditoraudit	ST_AUDITORAUDIT	None.
system.auditid	Not an attribute. Equivalent to field ST_AUDITID in BPXYSTAT.	ST_AUDITID	Read-only.
system.dmodelacl	Not an attribute. Equivalent to field.	ST_DMODELACL	Read-only.
system.fmodelacl	Not an attribute. Equivalent to field.	ST_FMODELACL	Read-only.
system.accessacl	Not an attribute. Equivalent to field.	ST_ACCESSACL	Read-only.
system.createtime	Not an attribute. Equivalent to field.	ST_CREATETIME	Read-only.
user.*	No equivalent.	No equivalent.	Needs specific PFS support.

## Related services

- [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory” on page 82](#)
- [“fchaudit \(BPX1FCA, BPX4FCA\) — Change audit flags for a file by descriptor” on page 199](#)
- [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor” on page 204](#)
- [“fchown \(BPX1FCO, BPX4FCO\) — Change the owner and group of a file or directory by descriptor” on page 210](#)
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor” on page 238](#)
- [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file” on page 249](#)
- [“lchattr \(BPX1LCR, BPX4LCR\) — Change the attributes of a file or directory or symbolic link” on page 371](#)

- [“truncate \(BPX1TRU, BPX4TRU\) — Change the size of a file ” on page 953](#)
- [“utime \(BPX1UTI, BPX4UTI\) — Set file access and modification times ” on page 981](#)

## Characteristics and restrictions

1. The ATTEXTLINK flag in the ATTGENVALUE field of BPXYATT cannot be modified with fchattr.
2. The General Attribute bits (set by ATTSETGEN, ATTGENMASK, and ATTGENVALUE fields) are not intended as a general-use programming interface to fchattr.
3. The security label (ATTSECLABELCHG) flag requires RACF SPECIAL authorization and appropriate privileges. It cannot be used to change an existing security label; it can only be used to set an initial security label on a file.

## Examples

See [“BPX1FCR \(fchattr\) example” on page 1237](#) and [“BPX4FCR \(fchattr\) example” on page 1331](#).

# fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor

---

## Function

The fchaudit callable service changes the types of access to a file to be audited for the security product. You identify the file by its file descriptor.

For the corresponding service using a path name, see [“chaudit \(BPX1CHA, BPX4CHA\) — Change audit flags for a file by path ” on page 95](#).

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FCA)</b>	31-bit
<b>AMODE (BPX4FCA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1FCA,(File_descriptor,
              Audit_flags,
              Option_code,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCA with the same parameters.

**Parameters****File\_descriptor**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword containing the file descriptor of the file to be changed.

**Audit\_flags**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword indicating the access to be audited. This field is mapped by the BPXYAUDT macro; see [“BPXYAUDT — Map flag values for chaudit and fchaudit” on page 1060](#). Values for this field include any combination of the following:

Value	Description
<b>AUDTREADFAIL</b>	Audit failing read requests.
<b>AUDTREADSUCCESS</b>	Audit successful read requests.
<b>AUDTWRITEFAIL</b>	Audit failing write requests.
<b>AUDTWRITESUCCESS</b>	Audit successful write requests.
<b>AUDTEXECFAIL</b>	Audit failing execute or search requests.
<b>AUDTEXECSUCCESS</b>	Audit successful execute or search requests.

**Option\_code**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that indicates whether you are changing the auditing for flags of the user or of the auditor. When this field has the value:

- 0: User audit flags are changed.
- 1: Auditor audit flags are changed.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fchaudit service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

**Return\_code****Explanation****EBADF**

The File\_descriptor parameter is not a valid file descriptor.

**EINVAL**

The Option\_code parameter is incorrect, or File\_descriptor refers to an unnamed pipe and fchaudit is not allowed on such a file.

**EPERM**The effective user ID of the calling process does not match the owner of the file, the calling process does not have appropriate privileges (see [“Authorization” on page 8](#)), or if Option\_code indicated that the auditor audit flags were to be changed, then the user may not have had auditor authority.**EROFS**

The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fchaudit service stores the reason code. The fchaudit service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If Option\_code indicates that the auditor audit flags are to be changed, the user must have auditor authority for the request to be successful. The user with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.  
You can get auditor authority by issuing the TSO/E command ALTUSER Auditor.
2. If Option\_code indicates that the user audit flags are to be changed, the user must have appropriate privileges (see [“Authorization” on page 8](#)), or be the owner of the file.

**Related services**

- [“chaudit \(BPX1CHA, BPX4CHA\) — Change audit flags for a file by path ” on page 95](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

**Characteristics and restrictions**

None.

**Examples**See [“BPX1FCA \(fchaudit\) example” on page 1236](#).

## fchdir (BPX1FCD, BPX4FCD) – Change the working directory

---

### Function

The fchdir service changes your working directory from the current one to a new one. The working directory is the starting point for path searches of path names not beginning with a slash.

For corresponding service using a path name, see [“chdir \(BPX1CHD, BPX4CHD\) – Change the working directory”](#) on page 98.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FCD)</b>	31-bit
<b>AMODE (BPX4FCD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1FCD, (Directory_file_descriptor,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FCD with the same parameters.

### Parameters

#### Directory\_file\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword containing the directory file descriptor that was returned when the directory was opened (see [“opendir \(BPX1OPD, BPX4OPD\) – Open a directory”](#) on page 523), which is to become the new working directory. It can also be specified as the name of a fullword containing the file descriptor of an open directory (see [“open \(BPX1OPN, BPX4OPN\) – Open a file”](#) on page 511).

#### Return\_value

Returned parameter

##### Type:

Integer



**Length:**

Fullword

The name of a fullword where the fchdir service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fchdir service stores the return code. The fchdir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fchdir service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EACCES	The calling process does not have search permission for the directory referenced by the file descriptor.
EBADF	The file descriptor parameter is not a valid file descriptor.
ENOTDIR	The open file descriptor does not refer to a directory. The following reason code can accompany the return code: JRChdNotDir.
EINTR	A signal was caught during the execution of fchdir().
EIO	An I/O error occurred while reading from or writing to the file system. The following reason codes can accompany the return code: JRQuiescing.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fchdir service stores the reason code. The fchdir service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“chdir \(BPX1CHD, BPX4CHD\) — Change the working directory” on page 98](#)
- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory” on page 124](#)
- [“getcwd \(BPX1GCW, BPX4GCW\) — Get the path name of the working directory” on page 260](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor” on page 423](#)
- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory” on page 523](#)
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory” on page 656](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory” on page 703](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

## Characteristics and restrictions

None.

## Examples

See “BPX1FCD (fchdir) example” on page 1237.

# fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor

---

## Function

The fchmod service modifies the permission bits used to control the owner access, group access, and general access to a file. It can be used to set flags that modify the user ID (UID) and group ID (GID) of the file when it is executed. It can also be used to set the sticky bit to indicate where the file should be fetched from. You identify the file by its file descriptor.

For the corresponding service using a path name, see “chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory ” on page 101.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FCM)</b>	31-bit
<b>AMODE (BPX4FCM)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1FCM, (File_descriptor,
               Mode,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FCM with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

### Type:

Integer

**Length:**

Fullword

Specifies the name of a fullword containing the file descriptor of the file whose mode you want to change.

**Mode**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

Specifies the name of a fullword in which the mode field is specified. The mode field, mapped by BPXYMODE, specifies the file type and the permissions you grant to yourself, to your group, and to any user. See [BPXYMOD](#) for the parameter options.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

Specifies the name of a fullword to which the fchmod service returns 0 if successful, or -1 if not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fchmod service stores the return code. The fchmod service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fchmod service can return one of the following values in the Return\_code parameter:

**EBADF**

The File\_descriptor parameter is not a valid file descriptor.

**EPERM**

The effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see [“Authorization”](#) on page 8).

**EROFS**

The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fchmod service stores the reason code. The fchmod service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. File descriptors open at the time of the call to the fchmod service retain the access permission they had at the time the file was opened.
2. For mode bits to be changed, the effective UID of the calling process must match the file's owner UID, or the process must have appropriate privileges (see [“Authorization” on page 8](#)).
3. When the mode is changed successfully, the file's change time is updated as well.
4. Setting the set-group-ID-on-execution permission means that when this file is run, through the exec call, the effective GID of the process is set to the file's owner GID, so that the process seems to be running under the GID of the file, rather than that of the actual invoker.

The set-group-ID-on-execution permission is suppressed (the bit is turned off) if both of the following are true:

- The calling process does not have appropriate privileges.
  - The file's owner GID does not match the effective GID or one of the supplementary GIDs of the calling process.
5. Setting the set-user-ID-on-execution permission means that when this file is run the process's effective UID will be set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

## Related services

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory” on page 101](#)
- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor” on page 423](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name” on page 893](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1FCM \(fchmod\) example” on page 1237](#).

# **fchmodat (BPX1CMA, BPX4CMA) — Change the mode of a file or directory relative to a directory file descriptor**

---

## Function

The fchmodat service modifies the permission bits that are used to control the owner access, group access, and general access to a file. Use it to set flags that modify the user ID (UID) and group ID (GID) of the file when it is executed. You can also use it to set the sticky bit to indicate from where the file should be fetched. You identify the file by its path name and the starting directory when a relative path name is specified.

For the corresponding service that uses a file descriptor, see [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor” on page 204](#).

Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1CMA)</b>	31-bit
<b>AMODE (BPX4CMA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1CMA,(Pathname_length,
               Pathname,
               Mode,
               Dirfd,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4CMA with the same parameters.

Parameters

Pathname\_length

Supplied parameter

Type

Integer

Length

Fullword

The name of a fullword that contains the length of the path name of the file whose mode you want to change.

Pathname

Supplied parameter

Type

Character string

Character set

No restriction

Length

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. This field has the length that is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory. When `Dirfd` is `AT_FDCWD`, the working directory is the current working directory. Otherwise, `Dirfd` contains the file descriptor for the starting directory.

**Mode**

Supplied parameter

**Type**

Structure

**Length**

Fullword

The name of a fullword that describes the access. This field, which is mapped by `BPXYMODE`, specifies the file type and permissions for the caller, for the callers group, and for any others. For more information, see [“BPXYMODE — Map the mode constants”](#) on page 1106.

**Dirfd**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains a file descriptor for the starting directory to be used for the change permissions request. The special value `AT_FDCWD` specified with a relative path name, indicates to use the current working directory to interpret the path. Otherwise, a relative path name is interpreted relative to the directory that is referred to by this file descriptor. If the path name is absolute, this parameter is ignored.

**Flags**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword field to further define the `fchmodat` request. Flags can either be 0 or include the following, which are defined in the `BPXYFCTL` macro.

**AT\_SYMLINK\_NOFOLLOW**

When `AT_SYMLINK_NOFOLLOW` is specified with a path name of a symbolic link, the permission change is performed on the link itself, as opposed to the default behavior where the link is dereferenced.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the `fchmodat` service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the fchmodat service stores the return code. The fchmodat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fchmodat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search some component of Pathname.
<b>EBADF</b>	Dirfd is not a valid file descriptor or AT_FDCWD.
<b>EINVAL</b>	An unsupported flag was specified (JrInvalidParms).
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters. Filename truncation is not supported.
<b>ENOENT</b>	No file that was named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Not a directory returned for the following reasons: <ul style="list-style-type: none"> <li>• Some component of Pathname is not a directory (JrCompNotDir).</li> <li>• The Dirfd parameter does not refer to a directory (JrNotDirFd).</li> </ul>
<b>EPERM</b>	The effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges. (See <a href="#">“Authorization”</a> on page 8.)
<b>EROFS</b>	Pathname specifies a file that is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the fchmodat service stores the reason code. The chmod service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. File descriptors that are open when the fchmodat service is called retain the access permission they had when the file was opened.
2. For mode bits to be changed, the effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges. (See [“Authorization”](#) on page 8.)
3. A user with READ authority to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class can use the chmod service to change the permission bits of any file.
4. When the mode is changed successfully, the file's change time is also updated.

5. Setting the set-group-ID-on-execution permission means that when this file is run, through the exec, spawn, or attach\_exec service, the effective GID of the caller is set to the file's owner GID so that the caller seems to be running under the GID of the file rather than the GID of the actual invoker.

The set-group-ID-on-execution permission is set to zero if both of the following are true:

- The caller does not have appropriate privileges.
- The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.

6. Setting the set-user-ID-on-execution permission means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than the UID of the actual invoker.

## Related services

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor” on page 204](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1CMA \(fchmodat\) example” on page 1227](#) and [“BPX4CMA \(fchmodat\) example” on page 1321](#).

# fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor

---

## Function

The fchown callable service changes the owner, group, or both owner and group of a file. You identify the file by its file descriptor.

For the corresponding service using a pathname, see [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#).

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FCO)</b>	31-bit
<b>AMODE (BPX4FCO)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts



<b>Operation</b>	<b>Environment</b>
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1FCO,(File_descriptor,
              Owner_UID,
              Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCO with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor of the file for which you wish to change the owner, group, or both owner and group.

### Owner\_UID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that contains the new owner UID assigned to the file, or the present value or -1 if there is no change. This parameter must be specified.

### Group\_ID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that contains the new group ID (GID) to be assigned to the file, or the present value or -1 if there is no change. This parameter must be specified.

### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword to which the fchown service returns 0 if the request is successful, or -1 if it is unsuccessful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fchown service stores the return code. The fchown service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fchown service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The File_descriptor parameter is not a valid file descriptor.
<b>EINVAL</b>	The Owner_UID or Group_ID parameter is incorrect; or File_descriptor refers to an unnamed pipe, and fchown is not allowed on such a file.
<b>EPERM</b>	The calling process does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).
<b>EROFS</b>	The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fchown service stores the reason code. The fchown service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The fchown service changes the owner UID and owner GID of a file. Only a process with appropriate privileges (see [“Authorization”](#) on page 8) can change the owner UID of a file.
2. The owner GID of a file can be changed by a process if the process has appropriate privileges, or if a process meets all of these conditions:
  - The effective UID of the process matches the file's owner UID.
  - The Owner\_UID value specified in the change request matches the file's owner UID.
  - The Group\_ID value specified in the change request is the effective GID, or one of the supplementary GIDs, of the calling process.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner\_UID and Group\_ID must be specified. If you want to change only one of these values, you must set the other to its present value or to -1 in order for it to remain unchanged.

**Related services**

- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory”](#) on page 104
- [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor”](#) on page 204

- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1FCO \(fchown\) example” on page 1237](#).

# fchownat (BPX1COA, BPX4COA) — Update the file ownership of a file or directory relative to a directory file descriptor

---

## Function

The fchownat service changes ownership of a file relative to a directory file descriptor.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1COA)</b>	31-bit
<b>AMODE (BPX4COA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1COA, (Pathname_length,
               Pathname
               Owner,
               Group,
               Dirfd,
               Flags
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4COA with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter.

### Type

Integer.

**Length**

Fullword.

Length of the path name.

**Pathname**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

The name of a field that contains the path name of the file. If the path name that is given in *Pathname* is relative, then it is interpreted relative to the directory referred to by the file descriptor *dirfd* (rather than relative to the current working directory of the calling process).

If the path name is relative and *dirfd* is the special value *AT\_FDCWD*, then *Pathname* is interpreted relative to the current working directory of the calling process.

If the path name is absolute, then *dirfd* is ignored.

**Owner**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The user ID type for the owner.

**Group**

Supplied parameter.

**Type.**

Integer

**Length.**

Fullword

The group ID type for the owner.

**Dirfd**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the directory file descriptor of the file whose ownership attribute that you want to change.

**Flags**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword field that indicates the *AT\_SYMLINK\_NOFOLLOW* option. This field is mapped by the *BPXYCHO* macro.

*flags* can either be 0, or include the following flag:

**AT\_SYMLINK\_NOFOLLOW**

If *pathname* is a symbolic link, do not dereference it. Instead, on the link itself, like **lchown**. By default, `fchownat()` dereferences symbolic links, like **chown**.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the `fchownat` service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the `fchownat` service stores the return code. The `fchownat` service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The same errors that occur for `chown` can also occur for `fchownat`. The following additional errors can occur for `fchownat`:

Return_code	Explanation
<b>EBADF</b>	<i>dirfd</i> is not a valid file descriptor.
<b>EINVAL</b>	An invalid flag value was specified in <i>flags</i> .
<b>ENOTDIR</b>	<i>pathname</i> is relative and <i>dirfd</i> is a file descriptor that refers to a file other than a directory.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the `chown` service stores the reason code. The `chown` service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The `fchownat` service changes the owner UID and owner GID of a file. Only a caller with appropriate privileges can change the owner UID of a file. For more information about appropriate privileges, see [“Authorization”](#) on page 8.
2. The owner GID of a file can be changed by a caller if the caller has appropriate privileges, or if a caller meets all of these conditions:
  - The effective UID of the caller matches the file's owner UID.
  - The `Owner_UID` value that is specified in the change request matches the file's owner UID.
  - The `Group_ID` value that is specified in the change request is the effective GID or one of the supplementary GIDs of the caller.

3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner\_UID and Group\_ID must be specified. To change only one of these values, set the one that is to remain unchanged to its present value or to -1.

**Related services**

- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“fchown \(BPX1FCO, BPX4FCO\) — Change the owner and group of a file or directory by descriptor” on page 210](#)
- [“lchown \(BPX1LCO, BPX4LCO\) — Change the owner or group of a file, directory, or symbolic link” on page 380](#)
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#)
- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

**Characteristics and restrictions**

None.

**Examples**

[“BPX1CHOA \(fchownat\) example” on page 1226](#) and [“BPX4CHOA \(fchownat\) example” on page 1320](#).

## **fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors**

---

**Function**

The fcntl callable service performs general control functions for open files: it retrieves or sets file descriptor flags, file status flags, locking information, and file tags. It also controls the automatic conversion of text data within files.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FCT)</b>	31-bit
<b>AMODE (BPX4FCT)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax is as follows:

```
CALL BPX1FCT,(File_descriptor,
              Action,
              Argument,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCT with the same parameters. However, for AMODE 64 callers, the Argument parameter may be either a 64-bit pointer or a 4-byte value, depending upon the Action parameter.

## Parameters

### File\_descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the file descriptor for the file. This parameter must specify an opened file descriptor, except when the Action parameter is F\_CLOSEFD, in which case this file descriptor is not expected to be in use.

### Action

Supplied parameter.

#### Type

Structure

#### Length

Fullword

The name of a fullword that contains an integer value mapped in the BPXYFCTL macro that indicates the action to be performed. For a list of actions, see [“BPXYFCTL — Command values and flags for fcntl”](#) on page 1076.

### Argument

Parameter supplied and returned

#### Type

Structure

#### Length

Fullword (doubleword)

The name of a fullword (doubleword) that contains an argument, or zero. The type of argument depends upon the action requested:

Action	Argument
F_CLOSEFD	File_descriptor_2
F_CONTROL_CVT	fcntl convert (F_CVT) structure
F_DUPFD	File_descriptor_2
F_DUPFD2	File_descriptor_2
F_DUP3	File_descriptor_2
F_GETFD	0
F_GETFL	0

Action	Argument
F_GETLK	Lock_information
F_GETOWN	0
F_SETFD	File_descriptor_flags
F_SETFL	File_status_flags
F_SETLK	Lock_information
F_SETLKW	Lock_information
F_SETOWN	Pid
F_SETTAG	File_Tag

For AMODE 64 callers that use F\_SETLK, F\_GETLK, F\_SETLKW, F\_SETTAG, or F\_CONTROL\_CVT, the Argument is a 64-bit pointer.

For AMODE 31 callers that use F\_SETLK, F\_GETLK, F\_SETLKW, F\_SETTAG, or F\_CONTROL\_CVT, the argument is a 31-bit pointer.

### Argument Options

#### File\_descriptor\_2

The name of a fullword that contains a file descriptor.

When Action is F\_DUPFD, fcntl returns the lowest file descriptor equal to or greater than File\_descriptor\_2 that is not already associated with an open file. File\_descriptor is duplicated.

When Action is F\_DUPFD2, the file descriptor that is returned is equal to File\_descriptor\_2. File\_descriptor\_2 is closed if it is already in use. F\_CLOEXEC is cleared.

File\_descriptor is duplicated. If File\_descriptor is equal to File\_descriptor\_2, the F\_DUPFD2 action returns File\_descriptor\_2 without closing it. F\_CLOEXEC is not cleared.

When Action is F\_CLOFD, File\_descriptor\_2 specifies the upper limit for the range of file descriptors to be closed, and File\_descriptor specifies the lower limit. If a -1 is specified for File\_descriptor\_2, all file descriptors greater than or equal to the lower limit are closed.

#### File\_descriptor\_flags

The name of a fullword that contains the file descriptor flags that are to be set or retrieved for File\_descriptor.

To get File\_descriptor\_flags, specify action F\_GETFD. If the action is successful, Return\_value maps to the bit settings of File\_descriptor\_flags.

Similarly, to set File\_descriptor\_flags, specify action F\_SETFD and use the mapping to set or reset File\_descriptor\_flags to the specified

**Restriction:** After the FCTLCLOFORK flag has been set on, it cannot be set off again.

File descriptor flags are mapped by the BPXYFCTL macro. For more information about the macro; see [“BPXYFCTL — Command values and flags for fcntl”](#) on page 1076.

#### File\_status\_flags

The name of a fullword that contains the file status flags to be set or retrieved for File\_descriptor.

To get File\_status\_flags, specify action F\_GETFL. If the action is successful, Return\_value maps to the bit settings of File\_status\_flags

Similarly, to set File\_status\_flags, specify action F\_SETFL and use the mapping to set or reset File\_status\_flags to the desired value. Only the O\_ASYNC, O\_APPEND, O\_NONBLOCK, and O\_SYNC flags are set when Action is F\_SETFL; any other flags specified are ignored.



File status flags are used to set some of the open flags that are mapped by the BPXYOPNF macro. For information about the mapping of the file status flags, see [“BPXYOPNF — Map flag values for open”](#) on page 1113.

Two masks are available for use with the return value from an F\_GETFL request. You can use the O\_ACCMODE mask to extract the file access mode flags from the return value. You can also use the O\_GETFL mask to extract both the file access mode and the file status flags.

### Lock\_information

The name of a fullword (doubleword) that contains a pointer to a structure that contains information about a file segment for which locks are to be set, cleared, or queried.

The Lock\_information is mapped by the BPXYBRLK macro as follows:

**0**

For **l\_type**, bytes 0-1 specify the type of lock that is being set, cleared, or queried. For more information, see "File Locking" in the usage notes.

**0**

For **l\_whence**, bytes 2-3 specify how the lock offset is to be determined. For more information, see "File Locking" in the usage notes.

**1-2**

**l\_start** specifies the starting byte offset of the lock that is to be set, cleared, or queried. This is a doubleword value.

**3-4**

**l\_len** specifies the length of the byte range that is to be set, cleared, or queried. This is a doubleword value.

**5**

For **l\_pid**, on return from a F\_GETLK request, this field contains the process ID of the process that is holding the blocking lock, if one was found.

For more information, see [“Usage notes for fcntl”](#) on page 223.

### Pid

The name of a fullword that contains either the process ID or the process group ID that is to receive the SIGIO or SIGURG signals for the socket that is associated with File\_descriptor.

Every socket has an associated process group number, which is initialized to zero. You set it by calling the fcntl service and specifying the F\_SETOWN action. This value can also be set using the w\_ioctl callable service. The Argument value for the F\_SETOWN can be a positive integer, specifying a process ID, or a negative integer (other than -1), specifying a process group ID. The F\_GETOWN command returns in the return value field either the process ID or the process group ID that is associated with the socket. The difference between specifying a process ID and specifying a process group ID is that in the first case only a single process receives the signal, while in the second case all processes in the process group receive the signal. The F\_SETOWN and F\_GETOWN actions are only available for AF\_INET stream sockets.

### File\_Tag

The name of a fullword (doubleword) that contains a pointer to a file tag. The file tag is mapped in BPXYSTAT ([“BPXYSTAT — Map the response structure for stat”](#) on page 1162).

When Action is F\_SETTAG, the fcntl service sets the file tag attributes for the file. The file must be a regular, FIFO, or character special file and must be opened in write mode. The file must be empty. If the file is not empty and the DeferTag bit is set, no error is returned and no processing occurs, assuming that the command would otherwise have worked. This allows the caller to issue F\_SETTAG without checking the file size, but not incur an error. If you use F\_SETTAG to set a tag that is already tagged and opened, O\_TRUNC is ignored.

When the DeferTag bit is off, the file tag is set immediately. When the DeferTag bit is on, the setting of the file tag is deferred until the first write by a call to BPX1WRT (BPX4WRT). The file tag is lost if no write ever occurs and the file is closed. If the write fails, file tagging might or might not have occurred. When the file is a FIFO or pipe, the file tag is deferred until the first read

(BPX1RED/BPX4RED) or first write (BPX1WRT/BPX4WRT), whichever comes first. This is because a read can precede a write when blocking is enabled, even for an empty file.

If the file is `/dev/null`, `/dev/random`, `/dev/urandom`, or `/dev/zero`, the file tag is not hardened to disk.

**Recommendation:** Using `F_SETTAG` multiple times with deferred tagging before the first write to the file is not recommended. Be aware that there are C-RTL environment options that may cause `F_SETTAG` with deferred tagging, such as `FILETAG(AUTOTAG)`.

### **fcntl convert structure**

The name of a two-word structure that describes how conversion is to occur for this file. The two-word structure is mapped in `BPXYFCTL` ([“BPXYFCTL — Command values and flags for fcntl” on page 1076](#); see `F_CVT`). The first word is one of four possible subcommands, followed by a 2-byte program CCSID and a 2-byte file CCSID.

When Action is `F_CONTROL_CVT`, the `fcntl` service controls how conversion occurs when the opened file is being read from (via `BPX1RED` or `BPX4RED`) or written to (via `BPX1WRT` or `BPX4WRT`). The file must be a regular, FIFO, character special file, or `AF_UNIX`.

The subcommands are:

#### **SetCvtAll**

Behaves the same as `SetCvtOn`, except automatic conversion is set to `ALL`, which enables UNICODE conversion. `ThliCcsid` is not used. `SetCvtAll` is ignored if I/O for the file has already started. A thread can set different program CCSIDs for each open file. However, an I/O error will result if any two threads have different program CCSIDs for the same open file that is shared by those two threads.

#### **SetCvtOff**

Turns off any conversion that may be in effect. The CCSID values are ignored. If automatic conversion is set to `ALL` and I/O has already started for the file, this command is ignored.

#### **SetCvtOn**

Turns automatic conversion on for the stream and, optionally, sets the program CCSID or file CCSID, or both. A hex value of 0 for `program-ccsid` indicates using the current program CCSID at the time of each read or write. The current program CCSID is initially 1047, but can be reset directly by the program, or indirectly by setting the appropriate runtime option or environment variable.

A hex value of 0 for the file CCSID indicates that the current setting is not to be changed. The values do not affect the stored file tag or program CCSID; they only change the values that are being used to control conversion on this data stream.

Setting or referencing `ThliCcsid` is still valid, but no longer recommended.

#### **SetAutoCvtAll**

If conversion is enabled for the environment (by `BPXPRMxx` `parmlib` statement `AUTOCVT` setting of `ALL` or with the appropriate environment variable), this subcommand behaves identically to `SetCvtAll`. Otherwise, it has no effect.

#### **SetAutoCvtOn**

If conversion is enabled for the environment (by `AUTOCVT` in `BPXPRMxx` or with the appropriate environment variable), this subcommand behaves identically to `SetCvtAll`. Otherwise, it has no effect.

#### **QueryCvt**

Returns information about whether conversion is in effect, and the program and file CCSIDs that are being used. On input, the subcommand is `QueryCvt`. On output, the subcommand is reset to `SetCvtOn`, `SetCvtAll`, or `SetCvtOff`, indicating that conversion is `ON` or `ALL` or `OFF`, respectively. The current CCSIDs are returned in their respective positions in the `F_CVT` structure.

### **Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the fcntl service returns 0 or greater, if the request is successful; or -1, if it is not successful. The following table lists the possible values of Return\_value for each action specified:

Action	Argument	Return_value
<b>F_CLOSEFD</b>	File_descriptor_2	0
<b>F_CONTROL_CVT</b>	F_CVT	0
<b>F_DUPFD</b>	File_descriptor_2	File_descriptor
<b>F_DUPFD2</b>	File_descriptor_2	File_descriptor
<b>F_DUPFD3</b>	File_descriptor_2	File_descriptor
<b>F_GETFD</b>	0	File_descriptor_flags
<b>F_GETFL</b>	0	File_status_flags
<b>F_GETLK</b>	Lock_information	Lock_information
<b>F_GETOWN</b>	0	Pid
<b>F_SETFD</b>	File_descriptor_flags	0
<b>F_SETFL</b>	File_status_flags	0
<b>F_SETLK</b>	Lock_information	0
<b>F_SETLKW</b>	Lock_information	0
<b>F_SETOWN</b>	Pid	0
<b>F_SETTAG</b>	File_Tag	0

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the fcntl service stores the return code. The fcntl service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fcntl service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	The calling process asked to set a lock, but the lock conflicts with a lock on an overlapping part of the file that is already set by another process.

Return_code	Explanation
<b>EBADF</b>	<p>The request was not accepted, for one of these reasons:</p> <ul style="list-style-type: none"> <li>• The File_descriptor parameter does not specify a valid, open file descriptor.</li> <li>• The request was to set a read lock, but the file is open for writing only.</li> <li>• The request was to set a write lock, but the file is open for reading only.</li> <li>• File_descriptor was opened with an opendir request. Many of the other requests are rejected for an opendir filedes.</li> <li>• If the action requested was F_DUPFD2, this error indicates that File_descriptor_2 was negative, or was equal to or greater than the highest file descriptor value allowed for the process. The MAXFILEPROC parmlib option is used to specify the largest file descriptor value for the system.</li> </ul> <p>The following reason code can accompany the return code: JRFDTooBig.</p>
<b>EDEADLK</b>	<p>The action requested was F_SETLKW; the potential for deadlock was detected.</p>
<b>EINTR</b>	<p>While processing a F_SETLKW request, fcntl was interrupted by a signal.</p>
<b>EINVAL</b>	<p>The request was not accepted, for one of these reasons:</p> <ul style="list-style-type: none"> <li>• If the action requested was F_DUPFD, File_descriptor_2 was negative, or it was equal to or greater than the highest file descriptor value that is allowed for the process. The MAXFILEPROC parmlib option is used to specify the largest file descriptor value for the system.</li> <li>• If the action requested was F_SETLK or F_SETLKW, the file that is specified by File_descriptor does not support locking, or the Lock_information parameter contains incorrect values.</li> <li>• The action requested was F_CLOSEFD and the file descriptor that is specified by File_descriptor_2 was less than File_descriptor, but not equal to -1.</li> <li>• The action requested was F_SETTAG or F_CONTROL_CVT, and either incorrect input data was supplied, or the file was inappropriate for this use.</li> <li>• An incorrect action was requested.</li> </ul> <p>The following reason codes can accompany the return code: JRFDTooBig, JRFD2TooSmall, JrBrImBadFileType, JrBrImBadL_Type, JrBrImInvalidRange, JrBrImBadL_Whence, JrNotSupportedForFileType, JrBadInputBufAddr, JrFileNotEmpty, JrWFildeRdOnly, JrInvalidFileTag, JrInvalidCcsid, JrBadOptCode.</p>
<b>EMFILE</b>	<p>The action requested was F_DUPFD. The process has already reached its maximum number of file descriptors, or there is no file descriptor available greater than File_descriptor_2.</p>
<b>ENOTSOCK</b>	<p>Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.</p>
<b>EPERM</b>	<p>The action requested was F_CLOSEFD, and at least one of the file descriptors in the specified range remains open. For a description of the file descriptors that cannot be closed with F_CLOSEFD, see <a href="#">“Usage notes for fcntl”</a> on page 223.</p>
<b>Reason_code</b> Returned parameter	

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the fcntl service stores the reason code. The fcntl service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes for fcntl**

When closing files:

- A process can use the fcntl service to close a range of file descriptors. File\_descriptor\_2 must be greater than or equal to File\_descriptor, or it can be -1, which indicates that all file descriptors greater than or equal to File\_descriptor are to be closed.
- Use of F\_CLOSEFD is meant to be consistent with use of the close service (BPX1CLO, BPX4CLO). You cannot close file descriptors that could not also be closed using the close service.
- If a file descriptor cannot be closed, it is considered an error, but the request continues with the next file descriptor in the range. File descriptors that are not in use are ignored.

When locking files:

- A process can use the fcntl service to lock out other cooperating processes from part of a file, so that the process can read or write to that part of the file without interference from others. This ensures data integrity when several processes are accessing a file concurrently.
- File locking can only be performed on file descriptors that refer to regular files. Locking is not permitted on file descriptors that refer to directories, FIFO files, pipes, character special files, or any other type of file.
- Locking operations are controlled with a structure mapped by BPXYBRLK, whose format is described in “Parameters” on page 217. This structure is needed whether the request is for setting a lock, releasing a lock, or querying a particular byte range for a lock. The following is a more detailed description of the BPXYBRLK structure.
- The l\_type field is used to specify the type of lock that is to be set, cleared, or queried. Valid values for l\_type are as follows:

**F\_RDLCK**

A *read lock*. Specified as a halfword integer value of 1, this is also known as a *shared lock*. This type of lock specifies that the process can read the locked part of the file, and other processes cannot write on that part of the file while it is doing so. A process can change a held write lock, or any part of it, to a read lock, thereby making it available for other processes to read. Multiple processes can have read locks on the same part of a file simultaneously. To establish a read lock, a process must have the file that is accessed for reading.

**F\_WRLCK**

A *write lock*. Specified as a halfword integer value of 2, this is also known as an *exclusive lock*. This type of lock indicates that the process can write on the locked part of the file, without interference from other processes. If one process puts a write lock on part of a file, no other process can establish a read lock or write lock on that same part of the file. A process cannot put a write lock on part of a file if there is already a read lock on an overlapping part of the file, unless that process is the only owner of that overlapping read lock. In such a case, the read lock on the overlapping section is replaced by the write lock that is being requested. To establish a write lock, a process must have the file that is accessed for writing.

**F\_UNLCK**

Indicates unlock. Specified as a halfword integer value of 3, this is used to unlock all locks held on the given range by the requesting process.

- The use of the `l_whence` and `l_start` fields for the `fcntl` service parallels their processing for the `lseek` service (BPX1LSK, BPX4LSK). See [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset ” on page 406](#) for more information.
- The `l_whence` field is used to specify how the byte range offset is to be found within the file. Valid values for `l_whence` are as follows:

**SEEK\_SET**

Stands for the start of the file, and is specified as a halfword integer value of 0.

**SEEK\_CUR**

Stands for the current file offset in the file, and is specified as a halfword integer value of 1.

**SEEK\_END**

Stands for the end of the file, and is specified as a halfword integer value of 2.

- The `l_start` field is used to identify the part of the file that is to be locked, unlocked, or queried. The part of the file that is affected by the lock begins at this offset from the location specified by the `l_whence` field. For example, if `l_whence` is `SEEK_CUR` and `l_start` is the value 10, a `F_SETLK` request attempts to set a lock beginning 10 bytes past the current cursor position. The `l_start` value may be negative, provided that when it is added to the offset indicated by the `l_whence` position, the resulting offset does not extend beyond the beginning of the file.

**Note:** Although you cannot request a byte range that begins or extends beyond the beginning of the file, you can request a byte range that starts or extends beyond the end of the file.

- The `l_len` field is used to give the size of the locked part of the file, in bytes. The value specified for `l_len` may be negative. If `l_len` is positive, the area affected begins at `l_start` and ends at `l_start+l_len-1`. If `l_len` is negative, the area affected begins at `l_start+l_len` and ends at `l_start-1`. If `l_len` is zero, the locked part of the file begins at the position specified by `l_whence` and `l_start`, and extends to the end of the file.
- The `l_pid` field identifies the process ID of the process that holds the lock found on an `F_GETLK` request, if one was found.

When obtaining locks:

- You can set locks by specifying `F_SETLK` as the Action parameter for the `BPX1FCT` (`BPX4FCT`) service. If the lock cannot be obtained, a `Return_value` of -1 is returned along with an appropriate `Return_code` and `Reason_code`. You can also use `F_SETLK` to release locks that are already held, by setting `l_type` to `F_UNLCK`.
- You can also set locks by specifying `F_SETLKW` as the Action parameter for the `BPX1FCT` (`BPX4FCT`) service. If the lock cannot be obtained because another process has a lock on all or part of the requested range, the `F_SETLKW` request waits until the specified range becomes free and the request can be completed. You can also use `F_SETLKW` to release locks that are already held, by setting `l_type` to `F_UNLCK`.
- If a signal interrupts a call to the `fcntl` service while it is waiting in an `F_SETLKW` operation, the function returns with a `Return_value` of -1 and a `Return_code` of `EINTR`.
- `F_SETLKW` operations can encounter deadlocks. This happens when process A is waiting for process B to unlock a region, and process B is waiting for process A to unlock a different region. If the system detects that an `F_SETLKW` might cause a deadlock, the `fcntl` service returns with a `Return_value` of -1 and a `Return_code` of `EDEADLK`.

When determining lock status:

- A process can determine locking information about a file by using `F_GETLK` as the Action parameter for the `fcntl` service. In this case, Argument should specify a pointer to a structure that is mapped by the `BPXYBRLK` macro. This structure should describe a lock operation that the caller would like to perform. When the `fcntl` service returns, the structure is modified to describe the first lock found that would prevent the proposed lock operation from completing successfully.
- If a lock is found that would prevent the proposed lock from being set, the `F_GETLK` request returns a modified structure whose:
  - The **`l_whence`** value is always `SEEK_SET`.

- The **l\_start** value gives the offset of the locked portion from the beginning of the file.
- The **l\_len** value is set to the length of the locked portion of the file.
- The **l\_pid** value is set to the process ID of the process that is holding the lock.

If there are no locks that would prevent the proposed lock operation from completing successfully, the returned structure is modified to have an **l\_type** of **F\_UNLCK**, but otherwise remains unchanged.

When there are multiple lock requests:

- A process can have several locks on a file simultaneously, but it can have only one type of lock set on any given byte. If a process puts a new lock on part of a file that it has previously locked, the process has only one lock on that part of the file and the lock type is the one given by the most recent locking operation.

When releasing locks:

- If an **F\_SETLK** or **F\_SETLKW** request is made to unlock a byte region of a file, all locks that are held by that process within the specified region are released. In other words, each byte specified on an unlock request is freed from any lock that is held against it by the requesting process.
- All of a process's locks on a file are removed when the process closes a file descriptor for that file. Locks are not inherited by a child process created with the **fork** service. See [“fork \(BPX1FRK, BPX4FRK\) — Create a new process”](#) on page 227 for more information about the **fork** service.

**Restriction:** All locks are advisory only. Processes can use locks to inform each other that they want to protect parts of a file, but locks do not prevent I/O on the locked parts. A process that has appropriate permissions on a file can perform any I/O it chooses, regardless of which locks are set. Therefore, file locking is only a convention, and it works only when all processes respect the convention.

## Related services

- [“close \(BPX1CLO, BPX4CLO\) — Close a file”](#) on page 121
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program”](#) on page 163
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process”](#) on page 227
- [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor”](#) on page 241
- [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset”](#) on page 406
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511

## Characteristics and restrictions

None.

## Examples

See [“BPX1FCT \(fcntl\) example”](#) on page 1238.

# flock (BPX1FLK, BPX4FLK) — Apply or remove an advisory lock on an open file

## Function

Use the **flock** service to apply or remove an advisory lock on an open file that is specified by a file descriptor, *fd*.

For the macro, see [“BPXYFLK— Applying or removing advisory locks”](#) on page 1078.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FLK)</b>	31-bit
<b>AMODE (BPX4FLK)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1FLK, (fd,
               Options,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FLK with the same parameters.

## Parameters

### fd

File descriptor for the open file.

### Options

LOCK\_SH

Places a shared lock. More than one process can hold a shared lock for a given file at a given time.

LOCK\_EX

Places an exclusive lock. Only one process can hold an exclusive lock for a given file at a given time.

LOCK\_UN

Removes an existing lock that is held by this process.

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

On success, 0 is returned. On error, -1 is returned and the return code is set.

### Return\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword



When the return\_value is -1, the following return codes are possible.

Return_code	Explanation
<b>EBADF</b>	<i>fd</i> is not an open file descriptor.
<b>EINTR</b>	During the wait to acquire a lock, the call was interrupted by delivery of a signal that was detected by a handler.
<b>EINVAL</b>	<i>operation</i> is invalid.
<b>ENOLCK</b>	The kernel ran out of memory for allocating lock records.
<b>EWOULDBLOCK</b>	The file is locked and the LOCK_NB flag is selected.

#### Reason\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

A fullword in which the flock service stores the reason code. The reason code is set only if the return code is -1. Reason\_code further qualifies the Return\_value value.

### Usage notes

1. A call to flock() might block if an incompatible lock is held by another process. To make a nonblocking request, include LOCK\_NB (by ORing) with any of the preceding operations.
2. A single file cannot simultaneously have both shared and exclusive locks. Locks created by flock() are associated with an open file table entry, which means that duplicate file descriptors, such as those created by fork() or dup(), refer to the same lock. This lock can be modified or released by using any of these descriptors. The lock is released either by an explicit LOCK\_UN operation on any of these duplicate descriptors, or when all such descriptors were closed.
3. If a process uses open() or a similar callable service to obtain more than one descriptor for the same file, these descriptors are treated independently by flock(). An attempt to lock the file by using one of these file descriptors might be denied by a lock that the calling process has already placed via another descriptor.
4. A process can only hold one type of lock (shared or exclusive) on a file. Subsequent flock() calls on an already locked file will convert an existing lock to the new lock mode.
5. Locks created by flock() are preserved across an execve().

### Related services

None.

### Characteristics and restrictions

None.

### Examples

See [“BPX1FLK \(flock\) example” on page 1239](#) and [“BPX4FLK \(flock\) example” on page 1332](#).

## fork (BPX1FRK, BPX4FRK) – Create a new process

### Function

The fork callable service creates a new process, called a *child process*.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, PSW key 8, TCB key 8.
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FRK)</b>	31-bit
<b>AMODE (BPX4FRK)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax is as follows:

```
CALL BPX1FRK, (Process_ID,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FRK with the same parameters.

## Parameters

### Process\_ID

Returned parameter.

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the fork service places the process ID of the newly created child process, 0, or -1.

Upon successful completion, fork returns the process ID of the newly created child to the calling (parent) process.

Because the child is a duplicate, it contains the same service request to the fork service as the parent. Execution of the child begins with this fork service returning a process ID value of zero; the child then proceeds with normal execution.

If Process\_ID is returned as -1, no child process was created, for the reason shown by Return\_code.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the fork service stores the return code. The fork service returns Return\_code only if Process\_ID is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fork service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	<p>The resources required to let another process be created are not available now; or you have already reached the maximum number of processes you are allowed to run.</p> <p>The following reason codes can accompany the return code: JRForcExitRcChildNoStorage, JRForcExitRcParentBadEnv, JRForcExitRcParentNoRoom, JRForcNoAccess, JRForcNoResource, JRForcVsmListTooLarge, JRKernelReady, JRMaxChild, JRMaxProc, JRMaxUIDs, JRNoSecurityProduct, JRNotKey8, and JRWlmWonErr.</p>
<b>EINVAL</b>	The following reason code can accompany the return code: JRJsRacXtr.
<b>ENOMEM</b>	<p>The process requires more space than is available.</p> <p>The following reason code can accompany the return code: JrNsInitProcTerm, JrNamespaceNotFound.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fork service stores the reason code. The fork service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes for fork**

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes the fork service, the child process that is created inherits the privilege of the set-user-ID or set-group-ID program.
2. The new process (called the *child process*) is a duplicate of the process that calls the fork service (called the *parent process*), except for the following:
  - The child process has a unique process ID (PID) that does not match any active process group ID.
  - The child has a different parent process ID (namely, the process ID of the process that called the fork service).
  - The child has its own copy of the parent's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the parent.
  - If the file has its FCTLCLFORK flag set on, it is not inherited by the child process. This flag is set with the fcntl service. For more information, see [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216.
  - The child has its own copy of the parent's open directory streams. Each open directory stream in the child can share directory stream positioning with the corresponding directory stream of the parent.
  - The process and system utilization times for the child are set to zero.
  - Any file locks previously set by the parent are not inherited by the child.
  - The child process has no interval timers set (similar to the results of a call to the alarm service with Wait\_time specified as zero).
  - The child has no pending signals.
  - The child is created in the same namespace as the calling process, unless a prior UNSHARE CLONE\_NEWPID or SETNS CLONE\_NEWPID was issued by the calling process.

In other respects, for z/OS UNIX the child is identical to the parent.

3. The child process inherits all key 8 shared memory segments that are attached to the calling process. The internal values of the number of processes that are attached to each shared memory segment (`shm_nattch`) are incremented.  
BPX1FRK only supports the propagation of key 8 storage; therefore, the fork service does not propagate to the child any shared memory segments that reside in a storage key other than key 8.
4. If the calling address space uses the macro `IARV SERV` to capture storage, these pages are not copied to the child address space.
5. The semaphore adjustment values (`semadj`) are cleared in the child process.
6. PSW Key 2 mmap storage areas are not propagated to the child. Above the bar key 2 and key 8 mmap storage areas are propagated to the child.
7. For AMODE 64 callers, high-memory storage is copied to the child process in the following cases:
  - All storage that is obtained by an IARV64 request that was made by the forking thread is copied to the child process.
  - All storage that is obtained by an IARV64 request with a user token that contains zeros in bits 0-31 and the parent process's PID in bits 32-64 is copied to the child process. In the child process, the user token is changed to the value of the child process's PID in bits 32-64.
  - All storage that is obtained by an IARV64 request with a user token that contains zeros in bits 0-31 and a nonzero value that matches `ThliParentTkn` in bits 32-64 (when `ThliChildTkn` is nonzero) is copied to the child process. In the child process, the user token is changed to the value of `ThliChildTkn` (from the parent process). This value is also used to initialize `ThliParentTkn` on the child process.
  - All authorized storage that is obtained by an IARV64 request with a user token that contains zeros in bits 32-64 and the parent process's PID in bits 0-31 is copied to the child process. In the child process, the user token is changed to the value of the child process's PID in bits 0-31.
  - All authorized storage that is obtained by an IARV64 request with a user token that contains zeros in bits 32-64 and the value of `PSALAA` in bits 0-31 is copied to the child process. In the child process, the user token is changed to the value of the child process's `LAA` in bits 0-31.

The child process inherits the `MEMLIMIT` of the parent.

The child address space inherits the following address space attributes of the parent address space: region size and time limit.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ” on page 574](#)
- [“setrlimit \(BPX1SRL, BPX4SRL\) — Set resource limits ” on page 789](#)
- [“times \(BPX1TIM, BPX4TIM\) — Get process and child process times ” on page 950](#)
- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)

## Characteristics and restrictions

Following is a list of characteristics or restrictions for the fork call:

- The fork service can be requested from either an MVS or kernel address space.
- The fork service is supported from programs that are running in PSW key 8 only. An additional requirement is that the storage protection key value in the **TCBPKF** field of the task control block

(TCB) must be 8. The fork service from authorized or problem-state programs with a PSW key other than 8 or a **TCBPKF** value other than 8 is rejected with an error code.

- Only the following storage subpools are copied by fork: 0-127, 129-132, and 251-252.
- With the exception of subpool 252, which is all key-0 storage, only the caller's key-8 storage is copied to the child. For subpools that support multiple keys (that is, subpool 129 to subpool 132) only storage that is obtained with a key of 8 is copied.
- When the fork service is called from a single-process address space, all storage that was obtained by all the tasks in the calling job step in the given subpools are copied to the child address space.

When the fork service is called from a multiple-process address space, only storage that is obtained by the tasks in the calling process in the subpools identified previously are copied to the child address space.

- The child process always runs in problem program state key of 8, even when it is forked by an APF-authorized MVS process.
- One task (thread) and one request block (RB) are present in the child address space after the fork service request.

If the parent was single-task with multiple RBs, only a single RB is created in the child address space after the fork service request. If multiple tasks exist in the parent process, only the task issuing the fork service request is replicated. There is no serialization among the different tasks.

- The TCB address and the addresses of other MVS control blocks are likely to be different in the child.
- The fork service does not copy any system subpools or MVS control blocks from the parent to the child, except as noted.

For example, the task I/O table (TIOT) is not copied. This means that MVS data sets that were allocated in the parent are not allocated to the child, except for the propagated TASKLIB, STEPLIB, or JOBLIB DD data sets. Because user data in user subpools is copied, it is possible that some of those control blocks can point to system control blocks that are no longer present in the child.

As another example, a user's data control block (DCB) that was opened in the parent still appears as an opened DCB in the child, but the corresponding system control blocks pointed to by the DCB are not present in the child.

Only services that are specifically documented as supported can be used across the fork service.

- There is a limit on the total number of living or zombie children the parent can have at a time. This limit is set with the MAXPROCUSER parameter in a BPXPRMxx parmlib member. You can retrieve this count with the sysconf service (BPX1SYC, BPX4SYNC).

Although the child process resembles the parent process in many ways, it has specific differences from the parent process. Besides, the differences described in POSIX.1 (under fork), the following are some examples of elements in the parent process that are not propagated to the child process:

- Linkage stack. The caller can have a linkage stack, but the child does not inherit it. If the caller intends to do an exec service request in the child, the loss of the linkage stack is not a problem. It is a problem only if the child process executes a PR (Program Return) instruction that requires the linkage stack.
- Access list (that is, PASN-AL, DU-AL). The parent's access lists are not propagated to the child.
- Access registers are not propagated to the child, because the child process does not inherit the parent's access list, which would be needed to use the access registers.
- Virtual pages that were page-fixed in the parent are not page-fixed in the child.
- Dynamic resource managers (RESMGRs). Dynamic resource managers that were established for the parent are not propagated to the child.
- Any MVS files that were opened for the parent are not opened for the child process, except for the TASKLIB, STEPLIB, or JOBLIB DD data sets that were propagated from the parent process. Only z/OS UNIX files are opened in the child process.
- Mutexes and condition variables. Because ownership of mutexes and condition variables is on a single-thread basis, these attributes cannot be propagated on fork. Where a mutex or condition variables

exists, the thread that is created in the child has access to the shared memory and can use the mutex or condition variable. However, when it begins running, it will not own any mutexes or consume any condition variables.

## Examples

See [“BPX1FRK \(fork\) example” on page 1239](#).

## MVS-related information

1. Following is a list of services in the child that relate to the services done in the parent:

- GETMAIN or FREEMAIN, or STORAGE. If the parent process has issued a GETMAIN macro for a storage block, the child process can issue a FREEMAIN macro for the same storage block.
- LOAD or DELETE. If a problem state parent process issues a LOAD macro for a module, the child process can issue a DELETE macro to remove the module from storage. If the child process issues a LOAD macro for the same module that was loaded in the parent, the copied version of the module is used and the use count is incremented.

If a supervisor state parent process issues a LOAD macro for a module, the child process cannot issue a DELETE macro for the module, and it cannot use a LOAD macro to load a new copy of the module.

However, a LOAD macro for global storage is not reflected in the child; the child cannot issue a DELETE macro to remove a module that was loaded to a common storage by the parent.

- CSVQUERY. The EPTOKEN (entry point token) returned as OUTEPTKN on a CSVQUERY macro in the parent can be used by the child as the INEPTKN parameter on a CSVQUERY macro to refer to the same module.
- ESTAE. The child process can issue an ESTAE macro with a 0 parameter to delete an ESTAE routine established by the parent process.
- ESPIE. The child process can delete an ESPIE routine established by the parent process.

**Note:** No other MVS services are carried across fork. They can be freely used in either the parent process or the child process, as long as it is understood that the result of these services (if performed in the parent process) cannot be available to the child process.

2. The system propagates the contents directory-related information (including extent lists) for the job pack queue for the job step task that is related to the task issuing the fork call. It also propagates the information on all modules (whether private or in the LPA) that have been loaded by the task issuing the fork call.
3. The system propagates the current task's SPIE or ESPIE and STAE or ESTAE status to the child process.
  - STAE or ESTAE control blocks that represent the current RB are propagated to the child process. Control blocks that are associated with older RBs are not propagated, nor are STAI or ESTAI control blocks.
  - SPIE or ESPIE control blocks that represent the current RB are propagated to the child process. SPIE or ESPIE control blocks that are associated with older RBs are not propagated.
4. Security information from the parent's address space is propagated to the child's address space. As a result, the child has a security environment equivalent to that of the parent.
5. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the current task are propagated to the child's address space. This causes the child address space to have the same MVS program search order as the calling parent task.
6. The accounting information of the parent's address space is propagated to the child's address space. (See in [Managing accounting work in z/OS UNIX System Services Planning](#).)

If the ThliForkAcctg bit is set on in [“BPXYTHLI — Thread-level information” on page 1165](#), the fork service creates the child with the accounting data from the RACF WORKATTR of the user ID that is

associated with the last setuid call. If no setuid call has been performed, the accounting information from the parent is used. No error is returned to the caller.

7. The job name of the parent is propagated to the child and appended with a numeric value in the range of 1–9 if the job name is 7 characters or fewer. If the job name is 8 characters, the job name is propagated as is. When a job name is appended with a numeric value, the count wraps back to 1 when it exceeds 9.
8. If the calling parent task is in a workload management (WLM) enclave, the child is joined to the same WLM enclave. This allows WLM to manage the parent and child as one business unit of work entity for system accounting and management purposes.
9. z/OS UNIX sets a default message class of A for all forked or spawned processes. Unlike JES, z/OS UNIX does not have a method for accepting a user-supplied default message class, and a default had to be supplied to the converter interpreter. Message class A was chosen as the default for BPXAS initiators. There is no way to dynamically change this default value. The MSGCLASS for the job log (JESMSG LG, JESJCL, JESYSMSG) is set to class A before the fork or spawn that associates the process with the BPXAS initiator is begun.
10. The user syscall trace setting is propagated to the child process.
11. To propagate APF authorization (JSCBAUTH) from the parent process to newly created child processes, turn on the THLIPROPAUTH flag in the thread-level information (THLI) control block associated with the thread issuing the fork callable service. The default behavior is not to propagate APF authorization from parent to child.

## fpathconf (BPX1FPC, BPX4FPC) – Determine configurable path name variables

---

### Function

The fpathconf callable service determines the current values of a configurable limit or option (variable) that is associated with a file or directory.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1FPC):</b>	31-bit
<b>AMODE (BPX4FPC):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax is as follows:

```
CALL BPX1FPC,(File_descriptor,
              Name,
              Return_value,
```

Return\_code,  
Reason\_code)

AMODE 64 callers use BPX4FPC with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor of the file.

### Name

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates which configurable limit or option (variable) is to be returned in the Return\_value. Use the BPXYPFC macro to specify the path name variable you want returned. The following table shows the variables that can be returned:

Variable returned	Description
<b>PC_CHOWN_RESTRICTED</b>	The change ownership (chown) function is restricted to processes with appropriate privileges (see “Authorization” on page 8). The group ID (GID) of a file can be changed only to the effective group ID of the process, or to one of its supplementary group IDs.
<b>PC_LINK_MAX</b>	The maximum value of a file's link count.
<b>PC_MAX_CANON</b>	The maximum number of bytes in a terminal canonical input line.
<b>PC_MAX_INPUT</b>	The minimum number of bytes for which space will be available in a terminal input queue. This is the maximum number of bytes a portable application may require to be typed as input before it reads them.
<b>PC_NAME_MAX</b>	The maximum number of bytes in a file name (not a string length; the count excludes a terminating null).
<b>PC_NO_TRUNC</b>	Path name components longer than 255 bytes generate an error.
<b>PATH_MAX</b>	The maximum number of bytes in a path name (not a string length; the count excludes a terminating null).
<b>PIPE_BUF</b>	The maximum number of bytes that can be written atomically when writing to a pipe.
<b>_POSIX_VDISABLE</b>	Terminal special characters maintained by the system can be disabled using this character value. For information on querying and setting these special characters, see <a href="#">“tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal” on page 926</a> or <a href="#">“tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal” on page 937</a> .
<b>PC_ACL</b>	The security product supports access control lists.
<b>PC_ACL_ENTRIES_MAX</b>	The maximum number of entries that can be placed in an access control list for the specified file.

### Return\_value

Returned parameter



**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fpathconf service returns the current value of the Pathname variable that corresponds to the Name specified, or -1 if the request is not successful.

If the named Pathname variable does not have a limit for the specified file, Return\_value is set to -1 and the Return\_code and Reason\_code remain unchanged.

If PC\_CHOWN\_RESTRICTED is specified for Name, and PC\_CHOWN\_RESTRICTED is active, Return\_value is set to 1.

If PC\_CHOWN\_RESTRICTED is specified for Name, and PC\_CHOWN\_RESTRICTED is not active, Return\_value is set to 0.

If PC\_NO\_TRUNC is specified for Name, and PC\_NO\_TRUNC is active, Return\_value is set to 1.

If PC\_NO\_TRUNC is specified for Name, and PC\_NO\_TRUNC is not active, Return\_value is set to 0.

If PC\_ACL is specified for Name, and PC\_ACL is supported, Return\_value is set to 1.

If PC\_ACL is specified for Name, and PC\_ACL is not supported, Return\_value is set to 0.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fpathconf service stores the return code. The fpathconf service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*.

If the named Pathname variable does not have a limit for the specified file, Return\_value is -1 and Return\_code is unchanged. Otherwise, the fpathconf service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The File_descriptor argument is not a valid file descriptor.
<b>EINVAL</b>	Refer to the usage notes for situations in which this is returned.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fpathconf service stores the reason code. The fpathconf service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If Name refers to MAX\_CANON, MAX\_INPUT, or \_POSIX\_VDISABLE, the following applies:
  - If File\_descriptor does not refer to a terminal file, the function returns -1 in Return\_value and sets the Return\_code to EINVAL.

2. If Name refers to NAME\_MAX, PATH\_MAX, or \_POSIX\_NO\_TRUNC, the following applies:
  - If File\_descriptor does not refer to a directory, the function still returns the requested information using the parent directory of the specified file.
3. If Name refers to PC\_PIPE\_BUF, the following applies:
  - If File\_descriptor refers to a pipe or a FIFO, the value returned applies to the referred-to object itself. If File\_descriptor refers to a directory, the value returned applies to any FIFOs that exist or that can be created within the directory. If File\_descriptor refers to any other type of file, the function returns -1 in Return\_value and sets the Return\_code to EINVAL.
4. If Name refers to PC\_LINK\_MAX, the following applies:
  - If File\_descriptor refers to a directory, the value returned applies to the directory.

**Related services**

- [“pathconf \(BPX1PCF, BPX4PCF\) — Determine configurable path name variables using a path name” on page 535](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1FPC \(fpathconf\) example” on page 1239](#).

## **freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr\_Info structures**

---

**Function**

The freeaddrinfo callable service frees the Addr\_Info structures that are obtained by the getaddrinfo callable service ([“getaddrinfo \(BPX1GAI, BPX4GAI\) — Get the IP address and information for a service name or location” on page 251](#)).

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FAI)</b>	31-bit
<b>AMODE (BPX4FAI)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1FAI, (Addr_Info_Ptr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FAI with the same parameters.

## Parameters

### Addr\_Info\_Ptr

Supplied parameter.

#### Type

Pointer

#### Length

Fullword

The name of a fullword field that contains a pointer to an Addr\_Info structure or a linked list of Addr\_Info structures returned by the getaddrinfo callable service. See Addr\_Info – AddrInfo Data Structure in the EZBREHST assembler macro for more information about the format of this structure. The EZBREHST macro is shipped in the installation's MACLIB SMP/E DDEF location.

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the freeaddrinfo service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

### Return\_code

Returned parameter.

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the freeaddrinfo service stores the return code. The freeaddrinfo service returns Return\_code only if Return\_value is -1. For a complete list of possible return code values, see *z/OS Communications Server: IP and SNA Codes*. The freeaddrinfo service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAI_AGAIN</b>	The resolver address space has not been started. Try the request later.
<b>EAI_FAIL</b>	An unrecoverable error occurred.

### Reason\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the freeaddrinfo service stores the reason code. The freeaddrinfo service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For the reason codes, see *z/OS Communications Server: IP and SNA Codes*.

## Usage notes

1. The freeaddrinfo service supports a thread-safe environment.
2. The pointer that is returned in the Results\_Ptr parameter of the getaddrinfo callable service can be specified with the Addr\_Info\_Ptr parameter on the freeaddrinfo callable service.
3. When the Addr\_Info\_Ptr parameter points to a linked list of Addr\_Info structures, the linked list of Addr\_Info structures is freed with one invocation of the freeaddrinfo callable service.

## Related services

- [“getaddrinfo \(BPX1GAI, BPX4GAI\) — Get the IP address and information for a service name or location” on page 251](#)
- [“getnameinfo \(BPX1GNI, BPX4GNI\) — Get the host name and service name from a socket address” on page 291](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1FAI \(freeaddrinfo\) example” on page 1236](#).

# **fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor**

---

## Function

The fstat callable service obtains status information about a file. You identify the file by its file descriptor.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FST)</b>	31-bit
<b>AMODE (BPX4FST)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1FST,(File_descriptor,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FST with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the file descriptor for the file.

### Status\_area\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the area to which the fstat call returns Status\_area. To determine the value of Status\_area\_length, use the BPXYSTAT macro (see [“BPXYSTAT — Map the response structure for stat”](#) on page 1162).

### Status\_area

Parameter supplied and returned

#### Type

Structure

#### Length

The length of BPXYSTAT or Status\_area\_length, whichever is less.

The name of an area to which the fstat call returns the status information for the file. Status\_area is mapped by the BPXYSTAT macro. For information on the contents of this macro, see [“BPXYSTAT — Map the response structure for stat”](#) on page 1162.

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword where the fstat service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the fstat service stores the return code. The fstat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fstat service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBADF</b>	The File_descriptor parameter does not identify a known file.
<b>EINVAL</b>	Parameter error; for example, a zero-length buffer was passed. The following reason code can accompany the return code: JRBuffTooSmall.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the fstat service stores the reason code. The fstat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. All time fields in the Status\_area are in POSIX format.
2. The file mode field in the Status\_area is mapped by BPXYMODE, and the file type field within the mode area is mapped by BPXYFTYP. For information about these fields, see [“BPXYMODE — Map the mode constants”](#) on page 1106 and [“BPXYFTYP — File type definitions”](#) on page 1078.
3. When the mode of an open file is changed by using a service such as chmod(), an fstat() call reflects the change in mode. However, no change in access authorization is apparent when the file is accessed through a previously opened file descriptor.
4. If no security label (SECLABEL) exists for the file, the security label field in the Status\_area contains binary zeros.

**Related services**

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216
- [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor”](#) on page 241
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name”](#) on page 893

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1FST \(fstat\) example”](#) on page 1239.

## fstatat (BPX1FSA, BPX4FSA) – Get file status relative to a directory file descriptor

---

### Function

The fstatat service obtains file status for the file indicated by the provided file descriptor relative to the provided path name. The target must be identified by a file descriptor. The relative path can be identified by a path name to a file or directory, a link name to a file or directory, or a symbolic link.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1FSA)</b>	31-bit
<b>AMODE (BPX4FSA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1FSA, (Pathname_length,
               Pathname,
               Stat_length,
               Stat,
               Dirfd,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FSA with the same parameters.

### Parameters

#### Pathname\_length

Supplied parameter.

##### Type

Integer

##### Length

Fullword

The name of a fullword that contains the length of Pathname.

#### Pathname

Supplied parameter.

##### Type

Character string

## **fstatat (BPX1FSA, BPX4FSA)**

### **Character set**

No restriction

### **Length**

Specified by the Pathname\_length parameter.

The name of a field that contains the path name of the directory relative to which the provided file descriptor should be located.

### **Dirfd**

Supplied parameter.

### **Type**

Integer

### **Length**

Fullword

### **Flags**

Supplied parameter.

### **Type**

Integer

### **Length**

Fullword

The name of a fullword field to further define the fstatat request. Flags can either be 0 or include the following, which are defined in the BPXYFCTL macro.

### **AT\_SYMLINK\_NOFOLLOW**

If *path* names a symbolic link, the status of the symbolic link is returned.

### **Return\_value**

Returned parameter.

### **Type**

Integer

### **Length**

Fullword

The name of a fullword where the fstatat service returns 0 if the request is successful or -1 if it is not successful.

### **Return\_code**

Returned parameter.

### **Type**

Integer

### **Length**

Fullword

The name of a fullword in which the fstatat service stores the return code. The fstat service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The fstat service can return one of the following values in the Return\_code parameter:



Return_code	Explanation
<b>EACCES</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• The calling process does not have permission to search some component of the path name, or does not have write permission for the directory that contains the link that is to be removed.</li> <li>• The S_ISVTX flag is set for the parent directory of the file or directory that is to be removed, and the caller is not the owner of the file or directory, or of the parent directory; nor does the caller have appropriate privileges (see “Authorization” on page 8).</li> <li>• The access mode of the open file descriptor associated with <code>fd</code> is not <code>O_SEARCH</code> and the permissions of the underlying directory do not permit directory searches</li> </ul>
<b>EBADF</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• The file descriptor specified for <code>Dirfd</code> is incorrect .</li> <li>• The path argument does not specify an absolute path and the <code>fd</code> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching</li> </ul>
<b>EINVAL</b>	<p>One of the input parameters was not valid. The following reason codes can accompany the return code:</p> <p><b>JrInvalidParms</b> A null character was encountered in the path name or an invalid flag was specified in the <code>Flags</code> parameter.</p> <p><b>JrDotOrDotDot</b> dot (.) or dot-dot (..) was specified in the path name.</p>
<b>ENOTDIR</b>	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the path argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the <code>Name</code> argument. This error is issued if more than 24 symbolic links are detected in the resolution of <code>Name</code> .
<b>ENAMETOOLONG</b>	Name is longer than 1023 characters, or a component of the path name is longer than 255 characters. Filename truncation is not supported.
<b>ENOENT</b>	<p>Name was not found, or no name was specified. The following reason codes can accompany the return code:</p> <p><b>JrUnlNoEnt</b> The file to be unlinked does not exist.</p> <p><b>JrFileNotThere</b> The requested directory does not exist.</p>
<b>Reason_code</b>	Returned parameter
<b>Type</b>	Integer
<b>Length</b>	Fullword

The name of a fullword in which the fstat service stores the reason code. The fstat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. All time fields in the Status\_area are in POSIX format.
2. The file mode field in the Status\_area is mapped by BPXYMODE, and the file type field within the mode area is mapped by BPXYFTYP. For information about these fields, see [“BPXYMODE — Map the mode constants”](#) on page 1106 and [“BPXYFTYP — File type definitions”](#) on page 1078.
3. When the mode of an open file is changed by using a service such as chmod(), an fstatat() call reflects the change in mode. However, no change in access authorization is apparent when the file is accessed through a previously opened file descriptor.
4. If no security label (SECLABEL) exists for the file, the security label field in the Status\_area contains binary zeros.

## Related services

- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor”](#) on page 238
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name”](#) on page 893

## Characteristics and restrictions

None.

## Examples

See [“BPX1FSA \(fstatat\) example”](#) on page 1240 and [“BPX1FSA \(fstatat\) example”](#) on page 1333.

# fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status

---

## Function

The fstatvfs callable service obtains status information about a file system. The file system is specified by a file descriptor that refers to a file from the desired file system.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1FTV):</b>	31-bit
<b>AMODE (BPX4FTV):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1FTV, (File_descriptor,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FTV with the same parameters.

**Parameters****File\_descriptor**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor for the file.

**Status\_area\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the area to which the service returns status information.

**Status\_area**

Parameter supplied and returned

**Type:**

Structure

**Length:**

Specified by the Status\_area\_length parameter

The name of an area of length Status\_area\_length to which the service returns the status information for the file system. The BPXYSSTF macro maps this area. For information on this macro, see [“BPXYSSTF — Map response structure for file system status” on page 1161](#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fstatvfs service returns the length of the status written to the Status\_area if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fstatvfs service stores the return code. The fstatvfs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fstatvfs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	Information is temporarily unavailable. This can occur because the mount process for the file system is incomplete.
<b>EBADF</b>	The File_descriptor parameter does not specify a valid, open file descriptor.
<b>EINVAL</b>	Parameter error; for example, Status_area_length is too small. The following reason code can accompany the return code: JRBuffTooSmall.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the fstatvfs service stores the reason code. The fstatvfs service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the passed Status\_area\_length is not less than or equal to zero, it is not considered an error for the Status\_area\_length to be insufficient to hold the requested information. (In other words, future expansion is allowed for.) As much information as can fit is written to Status\_area, and this amount is returned.
2. The amount of valid data returned in the Status\_area is indicated by the Return\_value. This allows for differences in the release levels of z/OS UNIX and the physical file systems.

**Related services**

- [“statvfs \(BPX1STV, BPX4STV\) — Get the file system status ” on page 896](#)
- [“w\\_statvfs \(BPX1STF, BPX4STF\) — Get the file system status ” on page 1029](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1FTV \(fstatvfs\) example” on page 1240](#).

## fsync (BPX1FSY, BPX4FSY) – Write changes to permanent storage

### Function

The fsync callable service writes changes on the permanent storage device that holds the file.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1FSY):</b>	31-bit
<b>AMODE (BPX4FSY):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1FSY, (File_descriptor,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4FSY with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor of the file for which changes are to be written to permanent storage.

#### Return\_value

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword in which the fsync service returns 0 if the request is successful, or -1 if it is not successful.

#### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fsync service stores the return code. The fsync service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The fsync service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The File_descriptor parameter does not specify a valid, open file.
<b>EINVAL</b>	The file is not a regular file.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the fsync service stores the reason code. The fsync service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The fsync service causes all modified data in the specified file to be written to the permanent storage device that holds the file. On return from a successful call, all updates have been saved on the permanent storage device that holds the file.
2. If the file represented by the file\_descriptor was opened with synchronous updates specified, there is no need to use the fsync callable service, because each write causes all updates to be written to permanent storage.

**Related services**

- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

**Characteristics and restrictions**

The file identified by File\_descriptor must be open for writing when the fsync service is called.

When automatic conversion is enabled, a fsync operation cannot write to permanent storage an untranslated partial character that was cached by z/OS. Untranslated partial characters occur when translating multibyte characters sets and z/OS is given data that does not end on a character boundary. Untranslated partial characters are resolved during a subsequent write operation (BPX1WRT/BPX4WRT) when the remaining part of the character is supplied.

**Examples**

See [“BPX1FSY \(fsync\) example” on page 1240](#).

## ftruncate (BPX1FTR, BPX4FTR) – Change the size of a file

---

### Function

The ftruncate service changes the size of a file. The file is identified by its file descriptor.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1FTR):</b>	31-bit
<b>AMODE (BPX4FTR):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax is as follows:

```
CALL BPX1FTR,(File_descriptor,
              File_length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FTR with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor of the file whose size is to be changed.

#### File\_length

Supplied parameter

##### Type:

Integer

##### Length:

Doubleword

The name of a doubleword that contains the number of bytes the file is to contain after its size has been changed.

This field is a doubleword to accommodate large files. For normal processing with a singleword value, propagate the sign bit through the second word, so that the final doubleword value has a valid sign. The ftruncate service accepts only positive values.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the ftruncate service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the ftruncate service stores the return code. The ftruncate service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The ftruncate service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBADF</b>	The File_descriptor parameter does not specify a valid, open file.
<b>EINVAL</b>	The file is not a regular file; it is opened Read Only; or the File_length specified is negative. The following reason codes can accompany the return code: JRTrNegOffset, JRTrNotRegFile, and JRTrOpenedRO.
<b>EROFS</b>	The specified file is on a read-only file system. The following reason code can accompany the return code: JRTrMountedRO.
<b>EFBIG</b>	The File_length parameter is greater than the maximum file size limit for the process. The following reason code can accompany the return code: JRWriteBeyondLimit.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the ftruncate service stores the reason code. The ftruncate service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The ftruncate service changes the file size to File\_length bytes, beginning from the first byte of the file. If the file was originally larger than File\_length bytes, the data from File\_length to the original end of the file is removed. If the file was originally shorter than File\_length, bytes between the old and new lengths are read as zeros.
2. If File\_length is greater than the soft file size limit for the process, the request fails with EFBIG, and the SIGXFSZ signal is generated for the process.



3. Full blocks are returned to the file system so that they can be used again.
4. The file offset is not affected by an ftruncate request.

## Related services

- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“truncate \(BPX1TRU, BPX4TRU\) — Change the size of a file ” on page 953](#)

## Characteristics and restrictions

The specified file must be a regular file, open for writing.

## Examples

See [“BPX1FTR \(ftruncate\) example” on page 1240](#).

# getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location

---

## Function

The getaddrinfo callable service converts the name of a service location (for example, a host name) or a service name (for example, FTP) into a set of socket addresses and other associated information. This information can be used to open a socket and connect to, or to send a datagram to, the specified service. The TCP/IP Services resolver attempts to resolve the host name through a name server, if one is present, or through the local data sets.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GAI):</b>	31-bit
<b>AMODE (BPX4GAI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GAI, (Node_Name,
               Node_Name_Length,
               Service_Name,
               Service_Name_Length,
               Hints_Ptr,
               Results_Ptr,
               Canonical_Length,
               Return_value,
```

Return_code, Reason_code)
------------------------------

AMODE 64 callers use BPX4GAI with the same parameters. Hints\_Ptr and Results\_Ptr are doubleword pointer fields.

## Parameters

### Node\_Name

Supplied parameter

#### Type:

Character

#### Character set:

EBCDIC

#### Length:

Specified by Node\_Name\_Length

Node\_Name can be specified as one of the following character strings:

1. An EBCDIC character string, up to 255 characters long, set to the node name (host name) that is being queried.
2. An EBCDIC character string set to the IP address of the node (host) where the service resides.

Also, Node\_Name can include scope information in the form *host name%scope information* or *IP address%scope information*. The scope information can be an interface name or an interface index, and must be specified as an EBCDIC character string. The combined character string of host name and scope information cannot exceed 255 characters in length.

You must specify Node\_Name or Service\_Name, or both.

### Node\_Name\_Length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the Node\_Name parameter.

### Service\_Name

Supplied parameter

#### Type:

Character

#### Character set:

EBCDIC

#### Length:

Specified by Service\_Name\_Length

Service\_Name can be specified as one of the following:

1. An EBCDIC character string, up to 32 characters long, set to the service name that is being queried.
2. An EBCDIC character string set to the port number of the required service.

You must specify Node\_Name or Service\_Name, or both.

### Service\_Name\_Length

Supplied parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Service\_Name parameter.

**Hints\_Ptr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a field that contains a pointer to an input Addr\_Info structure. The following information can be specified in the input Addr\_Info:

- A set of flags (*ai\_flags*) for interpreting the request. The flag settings are as follows:  
AI\_PASSIVE, AI\_CANONNAMEOK, AI\_NUMERICHOST, AI\_NUMERICSERV, AI\_V4MAPPED, AI\_ALL,  
and AI\_ADDRCONFIG.
- The address family (*ai\_family*) that the caller expects to be returned by the resolver. Valid settings are AF\_UNSPEC, AF\_INET, and AF\_INET6.
- The socket type (*ai\_socktype*) that the caller can accept as a response.
- The protocol (*ai\_protocol*) that the caller can accept as a response.

All other fields in the Addr\_Info structure must be set to zero.

See Addr\_Info – AddrInfo Data Structure in the EZBREHST assembler macro for more information about the format of this structure. The EZBREHST macro is shipped in the installation's MACLIB SMP/E DDEF location.

If the Hints\_Ptr parameter is not specified (zero), the invocation is treated as if ai\_family=AF\_UNSPEC, ai\_socktype=0, ai\_protocol=0, and all the ai\_flags are specified as off.

**Results\_Ptr**

Returned parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a field that contains a pointer to an output Addr\_Info structure. If more than one address is returned, this field contains a linked list of output Addr\_Info structures. Each output Addr\_Info structure contains the following information:

- A set of flags (*ai\_flags*) for interpreting the address that is returned in this Addr\_Info structure. For output Addr\_Info structures, this value is unpredictable.
- The address family (*ai\_family*) for the address returned in this Addr\_Info structure.
- The socket type (*ai\_socktype*) for the address returned in this Addr\_Info structure.
- The protocol (*ai\_protocol*) for the address returned in this Addr\_Info structure.
- The length (*ai\_addrlen*) of the sock\_inet\_sockaddr or sock\_inet6\_sockaddr structure returned in the ai\_addr field.
- The canonical name (*ai\_canonname*) associated with the input Node\_Name, if this was requested using the input AI\_CANONNAMEOK flag. If more than one Addr\_Info structure is returned, the canonical name is supplied in only the first Addr\_Info structure.

The length of the canonical name is returned in the Canonical\_Length parameter. If no canonical name exists, this field contains the input value that was passed in the Node\_Name parameter. If AI\_CANONNAMEOK in the input Addr\_Info structure was zero, ai\_canonname in the output Addr\_Info structure is set to zero.

- The socket address (ai\_addr) returned by the resolver in this Addr\_Info structure, in the form of a sock\_inet\_sockaddr or sock\_inet6\_sockaddr address structure. The length of the address returned is supplied by ai\_addrlen.
- The next Addr\_Info structure (ai\_next) returned by the resolver. If this is the last Addr\_Info structure returned as part of the reply, this value is X'00000000'.

See Addr\_Info – AddrInfo Data Structure in EZBREHST for more information about the format of this structure.

**Canonical\_Length**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getaddrinfo service returns the length of the canonical name that was returned in the first Addr\_Info structure pointed to by the Results\_Ptr parameter.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getaddrinfo service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getaddrinfo service stores the return code. The getaddrinfo service returns Return\_code only if Return\_value is -1. For a complete list of possible return code values, see *z/OS Communications Server: IP and SNA Codes*. The getaddrinfo service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EAI_NONAME</b>	One of the following situations occurred: <ol style="list-style-type: none"> <li>1. The name does not resolve for the specified parameters.</li> <li>2. A Name or Service operand was not specified. At least one of the Name or Service operands must be specified.</li> </ol>
<b>EAI_AGAIN</b>	The name specified by the Node_Name or Service_Name parameter could not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried later.
<b>EAI_FAIL</b>	An unrecoverable error occurred.
<b>EAI_SOCKTYPE</b>	The intended socket type was not recognized.
<b>EAI_SERVICE</b>	The service that was passed was not recognized for the specified socket type.

Return_code	Explanation
<b>EAI_BADFLAGS</b>	The ai_flags parameter had an incorrect setting.
<b>EAI_FAMILY</b>	The ai_family parameter had an incorrect setting.
<b>EAI_MEMORY</b>	A memory allocation failure occurred during an attempt to acquire an Addr_Info structure.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getaddrinfo service stores the reason code. The getaddrinfo service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. See *z/OS Communications Server: IP and SNA Codes* for the reason codes.

**Usage notes**

1. When you specify Node\_Name as an EBCDIC IP address, you must use the conventional forms for expressing IPv4 and IPv6 addresses as text strings. For example, the IPv4 address 1.1.1.1 would be specified as F14BF14BF14BF1, and the IPv6 address 1:1:1:1:1:1:1:1 would be specified as F17AF17AF17AF17AF17AF17AF1.
2. When you specify the AI\_NUMERICHOST flag in the input Addr\_Info structure pointed to by the Hints\_Ptr parameter, Node\_Name must be an IP address specified as an EBCDIC character string.
3. When you specify Node\_Name as an IP address, the address returned in the different structures is in different formats:
  - The IP address returned in the ai\_canonname field of the first Addr\_Info structure pointed to by Results\_Ptr is in its EBCDIC format.
  - The IP address returned in the sock\_inet\_sockaddr or sock\_inet6\_sockaddr structure of each returned Addr\_Info structure pointed to by Results\_Ptr is in numeric form (hexadecimal).
4. When the AI\_NUMERICSERV flag is specified in the input Addr\_Info structure pointed to by the Hints\_Ptr parameter, Service\_Name must be a port number specified as an EBCDIC character string.
5. The getaddrinfo service supports a fully thread-safe environment. The Addr\_Info structure or structures are allocated by the resolver and returned to the invoking application. The storage is subsequently returned to the resolver task, to be freed by the freeaddrinfo service ([“freeaddrinfo \(BPX1FAI, BPX4FAI\) — Free Addr\\_Info structures” on page 236](#)). The storage for the Addr\_Info structures is allocated in the caller's TCB key, and can be accessed in any key. To free the Addr\_Info structures using the freeaddrinfo service, or change the contents of the structures, the application must be in their TCB key.
6. To get the most useful set of IP addresses available for the requested host name, applications that are enabled for IPv6 processing should specify AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG in the ai\_flags field; and AF\_UNSPEC for the ai\_family field in the input Addr\_Info structure pointed to by the Hints\_Ptr parameter. When the stack has IPv6 capability, requests that are coded with AF\_UNSPEC are treated as if the request is for AF\_INET6, and all addresses are returned using sock\_inet6\_sockaddr structures (with the IPv4 addresses mapped appropriately, based on the AI\_V4MAPPED setting). If there is no IPv6 capability, IPv4 addresses are returned in sock\_inet\_sockaddr structures. This frees the application, to some extent, from having to decide what format works for the stack.
7. Applications are encouraged to attempt all returned addresses, in order, when using the getaddrinfo results to open a socket and connect, or to send a datagram.
8. Scope information specified as part of the Node\_Name input parameter is resolved into a zone index value and is returned in all the sock\_inet6\_sockaddr structures that represent IPv6 link-local

addresses. Scope information is ignored for resolved IPv4 addresses and for IPv6 addresses that are not link-local scoped addresses.

9. These are the flag descriptions specified with the Addr\_Info\_Structure parameter:

Flag	Description
<b>AI_PASSIVE (X'00000001')</b>	<p>Specifies how to complete the NAME pointed to in Results_Ptr. If this flag is specified, the returned address information will be suitable for use in binding a socket for accepting incoming connections for the specified service (such as the bind syscall). If the Node_Name parameter is not specified, the IP address portion of the socket address structure pointed to in Results_Ptr will be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY for an IPv6 address.</p> <p>If this flag is not specified, the returned address information will be suitable for the connect syscall (for a connection-mode protocol); or for a connect, sendto, or sendmsg syscall (for a connectionless protocol). In this case, if the Node_Name parameter is not specified, the IP address portion of the socket address structure pointed to by Results_Ptr will be set to the default loopback address for an IPv4 address (127.0.0.0) or the default loopback address for an IPv6 address (::1).</p> <p>This flag is ignored if the Node_Name parameter is specified.</p>
<b>AI_CANONNAMEOK (X'00000002')</b>	If this flag is specified and the Node_Name parameter is also specified, the getaddrinfo call attempts to determine the canonical name that corresponds to the Node_Name.
<b>AI_NUMERICHOST (X'00000004')</b>	If this flag is specified, the Node_Name parameter must be an IP address specified in EBCDIC format, or an error (EAI_NONAME) is returned.
<b>AI_NUMERICSERV (X'00000008')</b>	If this flag is specified, the Service_Name parameter must be a port number specified in EBCDIC format, or an error (EAI_NONAME) is returned.

Flag	Description
<b>AI_V4MAPPED (X'00000010')</b>	<p>If this flag is specified when <i>ai_family</i>=AF_INET6 or AF_UNSPEC in the input Addr_Info structure that is pointed to by the Hints_Ptr parameter, then the caller will accept IPv4-mapped IPv6 addresses.</p> <ul style="list-style-type: none"> <li>When the <i>ai_family</i> is AF_INET6, a query for IPv4 addresses is made if the AI_ALL flag is specified or if no IPv6 addresses are found. Any IPv4 addresses that are found are returned as IPv4-mapped IPv6 addresses.</li> <li>When the <i>ai_family</i> is AF_UNSPEC, queries are made for both IPv6 and IPv4 addresses. If IPv4 addresses are found and if IPv6 is supported, the IPv4 addresses are returned as IPv4-mapped IPv6 addresses.</li> <li>Otherwise, this flag is ignored.</li> </ul>
<b>AI_ALL (X'00000020')</b>	<p>When the <i>ai_family</i> field has a value of AF_INET6 and AI_ALL is set, the AI_V4MAPPED flag must also be set to indicate that the caller will accept all addresses (IPv6 and IPv4-mapped IPv6 addresses).</p> <p>When the <i>ai_family</i> field has a value of AF_UNSPEC, AI_ALL is accepted but it has no impact on the processing. Whether AI_ALL is set or not, the caller accepts IPv6 addresses and either IPv4 or IPv4-mapped IPv6 addresses. A query is first made for IPv6 addresses and if it is successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, and any found are returned as IPv4 addresses (if AI_V4MAPPED was not set or the system does not support IPv6) or as IPv4-mapped IPv6 addresses (if AI_V4MAPPED was set and the system supports IPv6).</p> <p>If the <i>ai_family</i> field does not have the value of AF_INET6, or the value of AF_UNSPEC when the system supports IPv6, the flag is ignored. Otherwise, this flag is ignored.</p>
<b>AI_ADDRCONFIG (X'00000040')</b>	<p>If this flag is specified, a query for IPv6 on the Node_Name occurs if the resolver determines whether either of the following situation is true:</p> <ul style="list-style-type: none"> <li>If the system is IPv6 enabled and has at least one IPv6 interface, the resolver will make a query for IPv6 (AAAA or A6 DNS) records.</li> <li>If the system is IPv4 enabled and has at least one IPv4 interface, the resolver will make a query for IPv4 (A DNS) records. The loopback address is not considered in this case as a valid interface.</li> </ul>

## Related services

- “freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr\_Info structures” on page 236
- “getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address” on page 291

## Characteristics and restrictions

None.

## Examples

See “BPX1GAI (getaddrinfo) example ” on page 1241.

# getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier

---

## Function

The getclientid callable service obtains the calling program's identifier.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GCL):</b>	31-bit
<b>AMODE (BPX4GCL):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GCL,(FunctionCode,  
              Domain,  
              Clientid,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4GCL with the same parameters.

## Parameters

### FunctionCode

Supplied parameter



**Type:**

Integer

**Length:**

Fullword

Specify a 1 to have the caller's name and task identifiers returned in the Clientid parameter. Specify a 2 to have the caller's process id returned in the Clientid parameter.

**Domain**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the communications domain in which the sockets are to be given and taken. For more information about the values that are defined for this field, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151.

**Clientid**

Returned parameter

**Type:**

Structure

**Length:**

Length of BPXYCID

The name of a structure that is to be returned with information that identifies the calling program.

If the FunctionCode parameter is 1, the returned Clientid is filled in as follows:

**CIdDomain**

Input Domain

**CIdName**

Calling program's address space name, left-justified, and padded with blanks

**CIdTask**

Calling program's subtask identifier

**CIdReserved**

Binary zeros

If the FunctionCode parameter is not 1, the returned Clientid is filled in as follows:

**CIdDomain**

Input Domain

**CIdName**

A fullword of binary zeros followed by the calling program's process id

**CIdTask**

Blanks

**CIdReserved**

Binary zeros

For more information about the format of this field, see [“BPXYCID — Map the returning structure for getclientid\(\)”](#) on page 1061.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getclientid service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getclientid service stores the return code. The getclientid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getclientid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	Using the Clientid parameter as specified would result in an attempt to access storage outside the caller's address space.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the getclientid service stores the reason code. The getclientid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The Clientid output of getclientid is intended to be used as the input Clientid of the givesocket and takesocket services.
2. The output Clientid that is returned with an input FunctionCode of 2 provides optimal performance and integrity when used as the input Clientid on the givesocket and takesocket services. The input FunctionCode of 1 is only provided for existing applications that may have been using the output of getclientid for purposes other than as input on the givesocket or takesocket services.

**Related services**

- [“givesocket \(BPX1GIV, BPX4GIV\) — Give a socket to another program ” on page 332](#)
- [“takesocket \(BPX1TAK, BPX4TAK\) — Acquire a socket from another program ” on page 916](#)

**Characteristics and restrictions**

None.

## getcwd (BPX1GCW, BPX4GCW) — Get the path name of the working directory

---

**Function**

The getcwd callable service gets the path name of the working directory.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1GCW)</b>	31-bit
<b>AMODE (BPX4GCW)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GCW, (Buffer_length,
               Buffer,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GCW with the same parameters.

## Parameters

### Buffer\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the buffer to which the `getcwd` service returns the path name of the directory. `Buffer_length` must be large enough to accommodate the actual length of the path name plus one (for the terminating null).

### Buffer

Parameter supplied and returned

#### Type

Character string

#### Character set

No restrictions

#### Length

Specified by the `Buffer_length` parameter

The name of the buffer that is to hold the path name of the working directory.

### Return\_value

Returned parameter

#### Type

Integer

**Length**

Fullword

The name of a fullword in which the `getcwd` service returns the length of the path name that is in the buffer, if the request is successful; or -1, if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the `getcwd` service stores the return code. The `getcwd` service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The `getcwd` service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The process did not have permission to read or search a component of the working directory's path name.
<b>EINVAL</b>	<code>Buffer_length</code> was specified as zero. The following reason code can accompany the return code: JRBuflenInvalid.
<b>EIO</b>	An input/output error occurred.
<b>ENOENT</b>	A component of a path name does not exist. This will be returned if a component of the working directory path name was deleted.
<b>ERANGE</b>	The specified <code>Buffer_length</code> is less than the length of the path name of the working directory.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the `getcwd` service stores the reason code. The `getcwd` service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“chdir \(BPX1CHD, BPX4CHD\) — Change the working directory ” on page 98](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GCW \(getcwd\) example ” on page 1241.](#)

## getegid (BPX1GEG, BPX4GEG) – Get the effective group ID

### Function

The getegid callable service gets the effective group ID (GID) of the calling process.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GEG):</b>	31-bit
<b>AMODE (BPX4GEG):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1GEG, (Effective_group_ID)
```

AMODE 64 callers use BPX4GEG with the same parameter.

### Parameters

#### Effective\_group\_ID

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword to which the getegid service returns the effective group ID of the calling process.

### Usage notes

If this service fails, the process ends abnormally.

### Related services

- [“geteuid \(BPX1GEU, BPX4GEU\) – Get the effective user ID” on page 264](#)
- [“getgid \(BPX1GID, BPX4GID\) – Get the real group ID ” on page 265](#)
- [“getuid \(BPX1GUI, BPX4GUI\) – Get the real user ID ” on page 329](#)
- [“setegid \(BPX1SEG, BPX4SEG\) – Set the effective group ID ” on page 758](#)
- [“seteuid \(BPX1SEU, BPX4SEU\) – Set the effective user ID” on page 760](#)
- [“setgid \(BPX1SGI, BPX4SGI\) – Set the group ID” on page 762](#)

- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)

### Characteristics and restrictions

None.

### Examples

See [“BPX1GEG \(getegid\) example ” on page 1242.](#)

## geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID

---

### Function

The geteuid callable service gets the effective user ID (UID) of the calling process.

### Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1GEU)	31-bit
AMODE (BPX4GEU)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1GEU,(Effective_user_ID)
```

AMODE 64 callers use BPX4GEU with the same parameter.

### Parameters

#### Effective\_user\_ID

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the geteuid service places the effective user ID of the calling process.

### Usage notes

If this service fails, the process ends abnormally.

## Related services

- [“getuid \(BPX1GUI, BPX4GUI\) — Get the real user ID ” on page 329](#)
- [“seteuid \(BPX1SEU, BPX4SEU\) — Set the effective user ID” on page 760](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GEU \(geteuid\) example” on page 1243](#).

# getgid (BPX1GID, BPX4GID) — Get the real group ID

---

## Function

The getgid callable service gets the real group ID (GID) of the calling process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GID):</b>	31-bit
<b>AMODE (BPX4GID):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GID,(Real_group_ID)
```

AMODE 64 callers use BPX4GID with the same parameter.

## Parameters

### Real\_group\_ID

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getgid service returns the real group ID.

## Usage notes

If this service fails, the process ends abnormally.

## Related services

- [“getgid \(BPX1GEG, BPX4GEG\) — Get the effective group ID” on page 263](#)
- [“setgid \(BPX1SEG, BPX4SEG\) — Set the effective group ID ” on page 758](#)
- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID” on page 762](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GID \(getgid\) example” on page 1245](#).

# getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database

---

## Function

The getgrent callable service gets information about a group and its members. Each time you use the getgrent service, you get information about the next group entry in the group database.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GGE):</b>	31-bit
<b>AMODE (BPX4GGE):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GGE, (Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GGE with the same parameters. The address returned is a fullword (below the bar).



## Parameters

### Return\_value

Returned parameter

#### Type:

Address

#### Length:

Fullword

The name of a fullword in which the getgrent service returns an address, or 0. If no more group entries exist in the group database, or if an error is encountered, Return\_value is set to 0.

If an entry is found, Return\_value is set to the address of a data area mapped by the BPXYGIDS macro. The first area contains the fullword length of the group name, followed by the group name, padded with blanks. See [“BPXYGIDS — Map data returned for getgrnam and getgrpid”](#) on page 1080. The address returned is 31 bits for both AMODE=31 and AMODE 64 callers.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getgrent service stores the return code or 0. The getgrent service returns Return\_code only if Return\_value is 0. Return\_code is 0 when no more group entries exist in the database. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The getgrent service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EMVSSAF2ERR</b>	The system authorization facility (SAF) or RACF Get GMAP service had an error.
<b>EMVSSAFEXTRERR</b>	The SAF or RACF RACROUTE EXTRACT service had an error.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword where the getgrent service stores the reason code or 0. The getgrent service returns Reason\_code only if Return\_value is 0. Reason\_code is 0 when no more group entries exist in the database. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

The reason code in the case of EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	12	Internal error during RACF processing
<b>8</b>	16	Unable to establish recovery
<b>8</b>	20	The group is incompletely defined.

**Usage notes**

1. The getgrent service is intended to be used to search the group database sequentially. The first call to this service from a given task returns a pointer to the first group entry in the group database. Subsequent calls from the same task return a pointer to the next group entry found, until no more entries exist. At this point a null pointer is returned.
2. The setgrent service can be used to reset this sequential search. The next getgrent service used from the same task after a call to setgrent returns a pointer to the first group entry. The next getgrent service used after an end of file indication (a null pointer) has been returned also returns a pointer to the first group entry. The use of setgrent after end of file is therefore optional.
3. The return value points to data that may change or go away after the next getgrgid, getgrnam, or getgrent service request from that task. Each task manages its own storage separately. Move data to the program's storage if it is needed for future reference.
4. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.

**Related services**

- [“getgrgid \(BPX1GGI, BPX4GGI\) — Access the group database by ID” on page 268](#)
- [“getgrnam \(BPX1GGN, BPX4GGN\) — Access the group database by name” on page 271](#)
- [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name” on page 289](#)
- [“setgrent \(BPX1SGE, BPX4SGE\) — Reset the group database” on page 765](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GGE \(getgrent\) example” on page 1243](#).

## **getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID**

**Function**

The getgrgid callable service gets information about a group and its members. You specify the group by the group ID (GID).

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GGI):</b>	31-bit
<b>AMODE (BPX4GGI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1GGI, (Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GGI with the same parameters. The returned address is a fullword (below the bar).

### Parameters

#### Group\_ID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword containing the ID of the group you want information about.

#### Return\_value

Returned parameter

**Type:**

Address

**Length:**

Fullword

The name of a fullword in which the getgrgid service returns an address, or 0. If no entry for the specified group ID is found, Return\_value is set to 0. If an entry is found, Return\_value is set to the address of the BPXYGIDS macro. The first area contains the fullword length of the group name, followed by the group name padded with blanks. See [“BPXYGIDS — Map data returned for getgrnam and getgrpid”](#) on page 1080.

If an entry is found and function code ThliEP\_GidNameSet was set in ThliEP\_FunctionCode prior to making this call, then Return\_value is set to the address of the area in the BPXYTHLI where group name length and group name are set, as per the BPXYGIDS macro.

The address returned is 31 bits for both AMODE 31 and AMODE 64 callers.

#### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgrgid service stores the return code. The getgrgid service returns Return\_code only if Return\_value is 0. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getgrgid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EMVSSAFEXTRERR	The system authorization facility (SAF) RACROUTE EXTRACT service had an error.
EMVSSAF2ERR	The (SAF) Get GMAP service had an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgrgid service stores the reason code. The getgrgid service returns Reason\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

The reason code in the case of EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

<b>RACF return code</b>	<b>RACF reason code</b>	<b>Explanation</b>
<b>8</b>	4	If the search is by GID, the GID is not defined. If the search is by group name, the current group is not defined.
<b>8</b>	8	The group name is not defined.
<b>8</b>	12	There was an internal error during RACF processing.
<b>8</b>	16	Recovery could not be established.
<b>8</b>	20	The current group is incompletely defined.

**Usage notes**

1. The return value points to data that might change or go away after the next getgrgid, getgrnam, or getgrent service request from that task. Each task manages its own storage separately. Move data to the program's storage if it is needed for future reference.
2. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.
3. Performance degradation can occur if you use this service when Virtual Lookaside Facility (VLF) is not active. For more information, see [Tuning performance](#) and [Establishing UNIX security](#) in *z/OS UNIX System Services Planning*.
4. You can request to have just the group name of the specified GID returned instead of the group name and all of the members of the group. This will improve performance because just one group name lookup is done. To request this function, set the ThliEP\_FunctionCode field with function code ThliEP\_GIDNameSet prior to issuing the syscall. On successful return, the following values are set:
  - ThliEP\_GIDNameLen is set to the length of the returned group name; valid values are 1-8.
  - ThliEP\_GIDName is set to the group name of the specified GID.
  - ThliEP\_GIDLlen is always set to 4, the length of a GID.
  - ThliEP\_GID is the GID specified on input.
  - ThliEP\_GroupCount is always set to 0.

Return\_value is set to the output area. This is mapped by both the ThliExtendedGIDName structure in [“BPXYTHLI — Thread-level information”](#) on page 1165, and by [“BPXYGIDS — Map data returned for getgrnam and getgrpid”](#) on page 1080. If no entry for the specified group ID is found, Return\_value is set to 0.

The ThliEP\_FunctionCode is cleared prior to returning to the caller; so, the invoker will need to set the ThliEP\_FunctionCode before each invocation of the BPX1GGI/BPX4GGI service if the ThliEP\_GIDNameSet function is to be used.

Related services

- [“getgrent \(BPX1GGE, BPX4GGE\) — Sequentially access the group database ” on page 266](#)
- [“getgrnam \(BPX1GGN, BPX4GGN\) — Access the group database by name” on page 271](#)
- [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name” on page 289](#)

Characteristics and restrictions

None.

Examples

See [“BPX1GGI \(getgrgid\) example” on page 1243.](#)

getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name

---

Function

The getgrnam callable service gets information about a group and its members. You specify the group by name.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GGN):	31-bit
AMODE (BPX4GGN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1GGN, (Group_name_length,  
               Group_name,  
               Return_value,  
               Return_code,  
               Reason_code)
```

AMODE 64 callers use BPX4GGN with the same parameters. The address returned is a fullword (below the bar).

Parameters

**Group\_name\_length**  
Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of Group\_name.

**Group\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Group\_name\_length parameter

The name of the field that contains the name of the group you want information about.

**Return\_value**

Returned parameter

**Type:**

Address

**Length:**

Fullword

The name of a fullword where the getgrnam service returns an address, or 0. If no entry for the specified group name is found, or no GID is specified, Return\_value is set to 0. If an entry is found, Return\_value is set to the address of the BPXYGIDS macro structure. The first area contains the fullword length of the group name followed by the group name padded with blanks.

If an entry is found and function code ThliEP\_GidNameSet was set in ThliEP\_FunctionCode prior to making this call, then Return\_value is set to the address of the area in the BPXYTHLI where the GID is set, as per the BPXYGIDS macro.

The address returned is 31 bits for both AMODE 31 and AMODE 64 callers. See [“BPXYGIDS — Map data returned for getgrnam and getgrpid”](#) on page 1080.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgrnam service stores the return code. The getgrnam service returns Return\_code only if Return\_value is 0. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getgrnam service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	Group name length is not valid.
<b>EMVSSAFEXTRERR</b>	The system authorization facility (SAF) RACROUTE EXTRACT service had an error.
<b>EMVSSAF2ERR</b>	The SAF Get GMAP service had an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the getgrnam service stores the reason code. The getgrnam service returns Reason\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*. The reason code in the case of EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	If the search is by GID: the GID is not defined. If the search is by group name: The current group is not defined.
8	8	The group name is not defined.
8	12	There was an internal error during RACF processing.
8	16	Recovery could not be established.
8	20	The current group is incompletely defined.

**Usage notes**

1. The return values point to data that can change or go away after the next getgrgid, getgrnam, or getgrent call from that task. Each task manages its own storage separately. Move data to your own dynamic storage if you need it for future reference.
2. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.
3. You can request to have just the GID of the specified group name returned without any information about the other group members. This will improve performance because just one group name look-up is done. To request this function, set the ThliEP\_FunctionCode field with function code ThliEP\_GIDNameSet prior to issuing the syscall. On successful return, the following values are set:
  - ThliEP\_GIDNameLen is set to the length of the returned group name; valid values are 1-8.
  - ThliEP\_GIDName is set to the specified group name.
  - ThliEP\_GIDLen is always set to 4, the length of a GID.
  - ThliEP\_GID is the GID associated with the input group name.
  - ThliEP\_GroupCount is always set to 0.

Return\_value is set to the output area. This is mapped by both the ThliExtendedGIDName structure in “BPXYTHLI — Thread-level information” on page 1165, and by “BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 1080. If no entry for the specified group ID is found, Return\_value is set to 0.

The ThliEP\_FunctionCode is cleared prior to returning to the caller; so, the invoker will need to set the ThliEP\_FunctionCode before each invocation of the BPX1GGI/BPX4GGI service if the ThliEP\_GIDNameSet function is to be used.

**Related services**

- [“getgrent \(BPX1GGE, BPX4GGE\) — Sequentially access the group database”](#) on page 266
- [“getgrgid \(BPX1GGI, BPX4GGI\) — Access the group database by ID”](#) on page 268
- [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name”](#) on page 289

## Characteristics and restrictions

None.

## Examples

See “BPX1GGRN (getgrnam) example ” on page 1244.

# getgroups (BPX1GGR, BPX4GGR) – Get a list of supplementary group IDs

---

## Function

The getgroups callable service gets the number of supplementary group IDs (GIDs) for the calling process. It optionally gets a list of those supplementary group IDs.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GGR):</b>	31-bit
<b>AMODE (BPX4GGR):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GGR,(Group_ID_list_size,  
              Group_ID_list_pointer_address,  
              Number_of_group_IDs,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4GGR with the same parameters. The Group\_ID\_list\_pointer\_address parameter is a doubleword.

## Parameters

### Group\_ID\_list\_size

Supplied parameter

#### Type:

Integer

#### Length:

Fullword



The name of a fullword that specifies the number of fullword entries in the group ID list. This number must be at least as great as the total number of group IDs for the process, or must be 0.

If you specify 0, the program receives only a count of the actual number of group IDs for the calling process, and not a list of those IDs.

#### **Group\_ID\_list\_pointer\_address**

Supplied parameter

##### **Type:**

Address

##### **Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains a pointer to a storage area in which the getgroups service is to place the list of supplementary group IDs. If Group\_ID\_list\_size is specified as 0, Group\_ID\_list\_pointer\_address is ignored, and does not have to be set to a valid address. When the request is successful, the storage area is an array of fullwords, each containing a supplementary group ID for the calling process.

#### **Number\_of\_group\_IDs**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the getgroups service returns a number that represents a count of supplementary group IDs. A -1 is returned if an error is detected.

- If Group\_ID\_list\_size is specified as 0, the number is the total number of supplementary group IDs for the process.
- If Group\_ID\_list\_size is specified as greater than 0 and the request was successful, the number is the actual number of group IDs that were put into the area specified by Group\_ID\_list\_pointer\_address.

#### **Return\_code**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the getgroups service stores the return code. The getgroups service returns Return\_code only if Number\_of\_group\_IDs is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getgroups service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	The Group_ID_list_size parameter was greater than 0 but less than the number of supplementary group IDs.
<b>EMVSSAF2ERR</b>	System authorization facility (SAF) had an error.

#### **Reason\_code**

Returned parameter

##### **Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgroups service stores the reason code. The getgroups service returns Reason\_code only if Number\_of\_group\_IDs is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

The reason code in the case of EMVSSAF2ERR contains the Resource Access Control Facility (RACF) return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF GETGRPS service return and reason code values, see the following table:

<b>RACF Return code</b>	<b>RACF reason code</b>	<b>Explanation</b>
<b>8</b>	4	Group count is less than the number of supplemental groups
<b>8</b>	8	Invalid grouplist address
<b>8</b>	12	Internal error during RACF processing

**Related services**

- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID” on page 762](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GGR \(getgroups\) example” on page 1244](#).

## **getgroupsbyname (BPX1GUG, BPX4GUG) — Get a list of supplementary group IDs by user name**

---

**Function**

The getgroupsbyname service gets the number of supplementary group IDs (GIDs) and, optionally, gets a list of those supplementary group IDs for a specified user name.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GUG):</b>	31-bit
<b>AMODE (BPX4GUG):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1GUG,(User_name_length,
              User_name,
              Group_ID_list_size,
              Group_ID_list_pointer_address,
              Number_of_group_IDs,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GUG with the same parameters. The Group\_ID\_list\_pointer\_address parameter is a doubleword.

**Parameters****User\_name\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of User\_name.

**User\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the User\_name\_length parameter

The name of a field of length User\_name\_length that contains the name of the user that you want information about. The name is specified in the Resource Access Control Facility (RACF) command that defined the user to the system.

**Group\_ID\_list\_size**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that specifies the number of fullword entries in the group ID list. This number must be at least as great as the total number of group IDs for the process, or must be 0.

If you specify 0, the program receives only a count of the actual number of group IDs for the calling process, and not a list of those IDs.

**Group\_ID\_list\_pointer\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains a pointer to a storage area where the getgroupsbyname service is to place the list of supplementary group IDs. If Group\_ID\_list\_size is specified as 0, Group\_ID\_list\_pointer\_address is ignored, and does not have to be set to a valid address. When the request is successful, the storage is an array of fullwords, each containing a supplementary group ID for the calling process.

**Number\_of\_group\_IDs**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgroupsbyname service returns the number of supplementary group IDs. A -1 is returned if an error is detected.

- If Group\_ID\_list\_size is specified as 0, the number is the total number of supplementary group IDs for the process.
- If Group\_ID\_list\_size is specified as greater than 0 and the request is successful, the number is the actual number of group IDs that are put into the area specified by Group\_ID\_list\_pointer\_address.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgroupsbyname service stores the return code. It returns Return\_code only if Number\_of\_group\_IDs is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getgroupsbyname service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	The Group_ID_list_size parameter was greater than 0 but less than the number of supplementary group IDs; or the User_name or User_name_length fields were incorrect.
<b>EMVSSAF2ERR</b>	A system authorization facility (SAF) service had an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getgroupsbyname service stores the reason code. The getgroupsbyname service returns Reason\_code only if Number\_of\_group\_IDs is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR, the reason code contains the Resource Access Control Facility (RACF) return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF GETGNAME service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	Group count is less than the number of supplemental groups
8	8	Incorrect group list address
8	12	Internal error during RACF processing
8	16	Unable to establish recovery
8	20	Internal error verifying user ID. The user ID might have been revoked.
8	24	User ID is not defined to RACF

## Related services

- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID” on page 762](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GUG \(getgroupsbyname\) example” on page 1251](#).

# gethostbyaddr (BPX1GHA, BPX4GHA) Get the IP address and alias of a host name for the specified IP address

---

## Function

The gethostbyaddr callable service returns the alias names and the internet addresses of a host whose address is specified as input. The TCP/IP Services resolver tries to resolve the host address through a name server, if one is present. If a name server is not present, the resolver searches for the HOSTS.ADDRINFO data set (or /etc hosts data set) until a matching host address is found, or until an EOF marker is reached.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GHA):</b>	31-bit
<b>AMODE (BPX4GHA):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GHA, (Address,
               Address_length,
               Hostent_ptr,
               Domain,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GHA with the same parameters. Hostent\_ptr is a doubleword pointer field.

## Parameters

### Address

Supplied parameter

#### Type:

Hexadecimal string

#### Length:

Length specified by Address\_length

The name of a hexadecimal string that contains the IP address of the host being queried. This is a fullword field for IPv4 addresses. (IPv6 addresses are not supported.)

### Address\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the address that is being passed in the Address parameter. This is 4 for IPv4 addresses. No other addresses are currently supported.

### Hostent\_ptr

Returned parameter

#### Type:

Pointer

#### Length:

Fullword (doubleword)

The name of a field that contains a pointer to the Hostent structure. The Hostent structure contains the following fields:

#### h\_name

The address of the host name returned by the service. The host name is a variable-length field that is ended by X'00'.

#### h\_aliases

The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer X'00000000'. Each alias name is a variable-length field that is ended by X'00'.

#### h\_addrtype

The value 2, which signifies AF\_INET.

#### h\_length

The length of the host internet addresses pointed to by h\_addr\_list.

#### h\_addr\_list

The address of a list of addresses that point to the host internet addresses returned by this service. This list is ended by the pointer X'00000000'.

**Domain**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the numeric value of the domain for this query. Only the value of 2 (AF\_INET) is currently supported.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostbyaddr service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostbyaddr service stores the return code. The gethostbyaddr service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The gethostbyaddr service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>HOST_NOT_FOUND</b>	The host name specified by the Address parameter was not found.
<b>TRY_AGAIN</b>	The host address specified by the Address parameter could not be resolved within the configured time interval. The request can be retried later.
<b>NO_RECOVERY</b>	An unrecoverable error occurred.
<b>NO_DATA</b>	The requested Address parameter is valid, but it does not have a record at the name server.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostbyaddr service stores the reason code. The gethostbyaddr service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value.

Reason codes that are less than decimal 4096 are z/OS UNIX System Services return codes. See [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*. Reason codes that are greater

than decimal 4096 are returned by the resolver, and are described in [Resolver reason codes](#) in *z/OS Communications Server: IP and SNA Codes*.

An assembler macro (EZBREHST) that contains the hostent structure, gethostbyxxxx return codes, and reason codes is shipped in the installation's MACLIB SMP/E DDEF location.

## Related services

- [“gethostbyname \(BPX1GHN, BPX4GHN\) Get IP information for specified host domain names”](#) on page 282

## Characteristics and restrictions

The gethostbyaddr service does not support a fully reentrant environment. The Hostent structure that is returned is allocated at a task level. This area will be reused on subsequent gethostbyaddr calls. Therefore, within a task only one call can be occurring at a time. For example, if the mainline task has issued a gethostbyaddr call that has not completed, a signal handler that interrupts that thread's processing should not invoke the gethostbyaddr service.

## Examples

See [“BPX1GHA \(gethostbyaddr\) example”](#) on page 1244.

# gethostbyname (BPX1GHN, BPX4GHN) Get IP information for specified host domain names

---

## Function

The gethostbyname callable service returns the alias names and the internet addresses of a host whose domain name is specified as input. The TCP/IP Services resolver tries to resolve the name through a name server, if one is present. If a name server is not present, the resolver searches for the HOSTS.SITEINFO data set (or /etc hosts data set) until a matching host name is found, or until an EOF marker is reached.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GHN):</b>	31-bit
<b>AMODE (BPX4GHN):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GHN, (Name,
               Name_length,
```



```
Hostent_ptr,  
Return_value,  
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4GHN with the same parameters. Hostent\_ptr is a doubleword pointer field.

## Parameters

### Name

Supplied parameter

#### Type:

Character

#### Length:

Length specified by Name\_length

A string, up to 255 characters long, set to the host name that is being queried.

### Name\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the Name parameter.

### Hostent\_ptr

Returned parameter

#### Type:

Pointer

#### Length:

Fullword (doubleword)

The name of a field that contains a pointer to the Hostent structure. The Hostent structure contains the following fields:

#### h\_name

The address of the host name returned by the service. The host name is a variable-length field that is ended by X'00'.

#### h\_aliases

The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer X'00000000'. Each alias name is a variable-length field that is ended by X'00'.

#### h\_addrtype

The value 2, which signifies AF\_INET.

#### h\_length

The length of the host internet addresses pointed to by h\_addr\_list.

#### h\_addr\_list

The address of a list of addresses that point to the host internet addresses returned by this service. This list is ended by the pointer X'00000000'.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the gethostbyname service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostbyname service stores the return code. The gethostbyname service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The gethostbyname service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>HOST_NOT_FOUND</b>	The host name specified by the Name parameter was not found.
<b>TRY_AGAIN</b>	The host name specified by the Name parameter could not be resolved within the configured time interval. The request can be retried later.
<b>NO_RECOVERY</b>	An unrecoverable error occurred.
<b>NO_DATA</b>	The requested Name parameter is valid, but it does not have a record at the name server.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostbyname service stores the reason code. The gethostbyname service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value.

Reason codes that are less than decimal 4096 are z/OS UNIX System Services return codes. See Reason codes in *z/OS UNIX System Services Messages and Codes*. Reason codes that are greater than decimal 4096 are returned by the resolver, and are described in [Resolver reason codes in z/OS Communications Server: IP and SNA Codes](#).

An assembler macro (EZBREHST) that contains the hostent structure, gethostbyxxxx return codes, and reason codes is shipped in the installation's MACLIB SMP/E DDEF location.

**Related services**

- [“gethostbyaddr \(BPX1GHA, BPX4GHA\) Get the IP address and alias of a host name for the specified IP address” on page 279](#)

**Characteristics and restrictions**

The gethostbyname service does not support a fully reentrant environment. The Hostent structure that is returned is allocated at a task level, and is reused on subsequent gethostbyname calls. Therefore, at any time only one call can be occurring within a task. For example, if the mainline task has issued a gethostbyname call that has not completed, a signal handler that interrupts that thread's processing should not invoke the gethostbyname service.

The Hostent structure is freed when the task is terminated.

**Examples**

See “BPX1GHN (gethostbyname) example ” on page 1245.

## gethostid or gethostname (BPX1HST, BPX4HST) – Get ID or name information about a socket host

---

**Function**

The gethostid or gethostname callable service obtains the ID or the name of the socket host.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1HST):</b>	31-bit task or SRB mode
<b>AMODE (BPX4HST):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1HST, (Domain,
               Name_length,
               Name,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4HST with the same parameters.

**Parameters****Domain**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the number that represents a domain. See “BPXYSOCK – Map SOCKADDR structure and constants” on page 1151 for valid Domain values.

**Name\_length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length of Name. If this field is zero, the information that is returned is the host ID. If this field is nonzero, the value that is supplied is the maximum length of the host name that is to be returned.

On return, this field contains the length of the name that is returned, including the trailing null. The size of this field should be less than 4096 bytes (4 KB) in length.

**Name**

Returned parameter

**Type:**

Character

**Length:**

Length specified by Name\_length.

The name of a field that contains the host name on successful return, if the request was gethostname. This name is terminated by a null character if there is sufficient room in the buffer.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostid or gethostname service returns one of the following:

- The host ID, if a zero-length Name\_length is supplied.
- 0, if a nonzero Name\_length is supplied and the name is successfully returned.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostid or gethostname service stores the return code. The gethostid or gethostname service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The gethostid or gethostname service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>ENOENT</b>	The domain that was specified was found to be not active. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRDomainNotSupported.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the gethostid or gethostname service stores the reason code. The gethostid or gethostname service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Characteristics and restrictions

These functions work only for AF\_INET sockets, and not for AF\_UNIX.

## Examples

See “BPX1HST (gethostid or gethostname) example” on page 1252.

## Usage notes

1. For more information about programming considerations for SRB mode, see [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041.

# getitimer (BPX1GTR, BPX4GTR) – Get the value of the interval timer

---

## Function

The getitimer callable service stores the current value of the timer specified into a structure.

## Requirements

Operation	Environment
<b>Authorization:</b>	Problem Program or Supervisor State, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GTR):</b>	31-bit
<b>AMODE (BPX4GTR):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GTR, (Interval_Type,
               Interval_Value_Adr
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GTR with the same parameters. The Interval\_Value\_Adr parameter is a doubleword.

## Parameters

### Interval\_Type

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains a numeric value that identifies the interval timer and format of the structure that is pointed to by Interval\_Value\_Adr. This parameter can have the following values:

- ITIMER\_REAL = Real time (the default if VIRTUAL and PROF are not specified)
- ITIMER\_VIRTUAL = Virtual time (CPU time minus system time)
- ITIMER\_PROF = CPU time
- ITIMER\_MICRO = Initial and reload times are in microseconds (the default if NANO is not specified)
- ITIMER\_NANO = Initial and reload times are in nanoseconds

The ITIMER\_ constants are defined in the BPXYITIM macro.

### Interval\_Value\_Adr

Supplied parameter

**Type:**

address

**Length:**

Fullword (doubleword)

A fullword (doubleword) field that contains an address that points to a structure that is defined by the BPXYITIM macro. This structure contains the time remaining and reload values, in seconds and either microseconds or nanoseconds.

### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getitimer service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getitimer service stores the return code. The getitimer service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The getitimer service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value specified for Interval_Type is not valid. (JRIntervalTypeInvalid).

### Reason\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getitimer service stores the reason code. The getitimer service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The number of seconds that is returned is unsigned and may exceed the setitimer allowable limit. This can happen if alarm is set for up to X'FFFFFFFF' seconds.
2. In 31-bit mode, the first two words returned are seconds, and then micro or nanoseconds. In 64-bit mode, the first doubleword in the structure is seconds, the next word is reserved, and the next word is the micro or nanoseconds. Although the structure returned in 64-bit mode for seconds is a doubleword, the value is the same as if it were a single word.
3. The three interval timers are:
  - ITIMER\_REAL, which decrements in real time. A SIGALRM signal is delivered when this timer expires.
  - ITIMER\_VIRTUAL, which decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.
  - ITIMER\_PROF, which decrements both in process virtual time, and when the system is running on behalf of the process. A SIGPROF signal is delivered when it expires.
  - Nanosecond values are subject to rounding.
  - Reload values may be changed to a system-imposed minimum.

**MVS-related information**

- “setitimer (BPX1STR, BPX4STR) — Set the value of the interval timer ” on page 768
- “alarm (BPX1ALR, BPX4ALR) — Set an alarm ” on page 30

**Characteristics and restrictions**

None.

**Examples**

See “BPX1GTR (getitimer) example” on page 1251.

## getlogin (BPX1GLG, BPX4GLG) — Get the user login name

---

**Function**

The getlogin callable services gets the user login name that is associated with the current process.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN

Operation	Environment
AMODE (BPX1GLG):	31-bit
AMODE (BPX4GLG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1GLG,(Return_value)
```

AMODE 64 callers use BPX4GLG. The address returned is a fullword (below the bar).

Parameters

**Return\_value**  
Returned parameter

**Type:**  
Address

**Length:**  
Fullword

The name of a fullword to which the getlogin service returns a pointer to a login name field, or 0. If a login name is not found, Return\_value is set to 0. If a login name is found, Return\_value is set to the address of a field that contains the length of the login name and the login name. The address returned is 31 bits for both AMODE 31 and AMODE 64 callers. The login name length is a fullword. Batch processing has a user name that is associated with a process; this user name is used as the



Usage notes

If this service fails, the process ends abnormally.

Related services

- [“geteuid \(BPX1GEU, BPX4GEU\) — Get the effective user ID” on page 264](#)
- [“getpwnam \(BPX1GPN, BPX4GPN\) — Access the user database by user name ” on page 305](#)
- [“getpwuid \(BPX1GPU, BPX4GPU\) — Access the user database by user ID ” on page 308](#)
- [“getuid \(BPX1GUI, BPX4GUI\) — Get the real user ID ” on page 329](#)

Characteristics and restrictions

None.

Examples

See [“BPX1GLG \(getlogin\) example” on page 1246](#).



## getpeername or getsockname (BPX1GNM, BPX4GNM) – Get the name of a socket or of the peer connected to a socket

---

See “[getsockname or getpeername \(BPX1GNM, BPX4GNM\) - Get the name of a socket or connected peer](#)” on page 319.

## getnameinfo (BPX1GNI, BPX4GNI) – Get the host name and service name from a socket address

---

### Function

The getnameinfo callable service resolves a socket address into a host name and a service name. The TCP/IP Services resolver attempts to resolve the socket address through a name server, if one is present, or through the local data sets.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GNI):</b>	31-bit
<b>AMODE (BPX4GNI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1GNI,(SockAddr,
              SockAddr_Length,
              Service_Buffer,
              Service_Buffer_Length,
              Host_Buffer,
              Host_Buffer_Length,
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GNI with the same parameters.

### Parameters

#### SockAddr

Supplied parameter

#### Type:

Structure

**Length:**

Specified by SockAddr\_Length

The name of a field that contains the socket address to be resolved. The socket address consists of an address family, a port number, and an IP address. If the IP address is a link-local IPv6 address, the socket address can also contain a zone index field.

The IP address is resolved to a host name and returned in the Host\_Buffer parameter. The port number is resolved to a service name and returned in the Service\_Buffer parameter. The zone index field is resolved to an interface name and appended to the host name in the form hostname%scope information.

The format of SockAddr is determined by the domain in which the socket descriptor was created. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for additional information on the format of SockAddr.

**SockAddr\_Length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the SockAddr parameter.

**Service\_Buffer**

Supplied and returned parameter

**Type:**

Character

**Character set:**

EBCDIC

**Length:**

Specified by Service\_Buffer\_Length

The name of a field into which the service name, resolved from the port number that was specified as part of the SockAddr parameter, is returned as an EBCDIC string. The maximum length of the returned service name is 32 bytes. If the storage specified is inadequate to contain the resolved service name, the service name is returned only up to the specified storage, and truncation can occur.

**Service\_Buffer\_Length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Service\_Buffer parameter. Upon return from the getnameinfo service, Service\_Buffer\_Length contains the length of the name returned in the Service\_Buffer parameter.

If Service\_Buffer\_Length is zero, nothing is returned in Service\_Buffer.

**Host\_Buffer**

Supplied and returned parameter

**Type:**

Character

**Character set:**

EBCDIC

**Length:**

Specified by Node\_Buffer\_Length

The name of a field into which the host name, resolved from the IP address that was specified as part of the SockAddr parameter, is returned as an EBCDIC string. The maximum length of the returned host name is 255 bytes. If the storage specified is inadequate to contain the resolved host name, the host name is returned only up to the specified storage, and truncation can occur.

If the IP address specified as part of the SockAddr parameter represents a link-local IPv6 address and the zone index specified as part of the SockAddr parameter is nonzero, then the information returned includes scope information in the form *hostname%scope information*. When the NI\_NUMERICSCOPE flag is specified with the Flags parameter, the scope information returned is the zone index value in numeric form (EBCDIC decimal); otherwise, the scope information returned is the interface name associated with the zone index. The maximum length for the combined hostname and scope information remains 255 bytes.

If the NI\_NUMERICHOST flag is specified with the Flags parameter, or the host name cannot be located, the IP address, specified as part of the SockAddr parameter, is returned in Host\_Buffer in numeric form (EBCDIC decimal).

**Host\_Buffer\_Length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Host\_Buffer parameter.

Upon return from the getnameinfo service, Host\_Buffer\_Length contains the length of the name returned in the Host\_Buffer parameter.

If Host\_Buffer\_Length is zero, nothing is returned in Host\_Buffer.

**Flags**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains flags for controlling the resolution of the socket address.

Flag	Value	Description
<b>NI_NOFQDN</b>	X'00000001'	Only the host name portion of the FQDN is to be returned for local hosts.
<b>NI_NUMERICHOST</b>	X'00000002'	The numeric form of the host's address is to be returned, instead of its name.
<b>NI_NAMEREQD</b>	X'00000004'	If the host name cannot be located, an error or NULL character is to be returned.
<b>NI_NUMERICSERV</b>	X'00000008'	The numeric form of the service name is to be returned (its port number), instead of its name.

Flag	Value	Description
<b>NI_DGRAM</b>	X'00000010'	The service is a datagram service (SOCK_DGRAM). The default behavior is to assume that the service is a stream service.
<b>NI_NUMERICSCOPE</b>	X'00000020'	The numeric form of the scope information is to be returned (the zone index), rather than the interface name.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getnameinfo service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getnameinfo service stores the return code. The getnameinfo service returns Return\_code only if Return\_value is -1. For a complete list of possible return code values, see [Resolver reason codes](#) in *z/OS Communications Server: IP and SNA Codes*. The getnameinfo service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAI_NONAME</b>	The host name does not resolve for the supplied parameters. One of the following conditions occurred: <ol style="list-style-type: none"> <li>1. NI_NAMEREQD is set, and the host name cannot be located.</li> <li>2. Both host name and service name were null.</li> </ol>
<b>EAI_BADFLAGS</b>	The flags parameter had an incorrect value.
<b>EAI_FAMILY</b>	The address family was not recognized, or the address length was not valid for the specified family.
<b>EAI_MEMORY</b>	A memory allocation failure occurred.
<b>EAI_AGAIN</b>	The specified host address could not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried later.
<b>EAI_FAIL</b>	An unrecoverable error occurred.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getnameinfo service stores the reason code. The getnameinfo service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For the reason codes, see [Resolver reason codes](#) in *z/OS Communications Server: IP and SNA Codes*.

**Usage notes**

1. The getnameinfo service supports a fully thread-safe environment.
2. You must specify either Service\_Buffer and Service\_Buffer\_Length, or Host\_Buffer and Host\_Buffer\_Length.

**Related services**

- [“freeaddrinfo \(BPX1FAI, BPX4FAI\) — Free Addr\\_Info structures” on page 236](#)
- [“getaddrinfo \(BPX1GAI, BPX4GAI\) — Get the IP address and information for a service name or location” on page 251](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GNI \(getnameinfo\) example” on page 1246](#).

## getpgid (BPX1GEP, BPX4GEP) — Get the process group ID

---

**Function**

The getpgid callable service gets the process group ID of the process whose process ID is equal to the input process ID. If the input process ID is 0, getpgid returns the process group ID of the calling process.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GEP):</b>	31-bit
<b>AMODE (BPX4GEP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

## getpgid (BPX1GEP, BPX4GEP)

```
CALL BPX1GEP,(PID,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4GEP with the same parameter.

### Parameters

#### PID

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the process ID for which to get the process group ID. If PID is 0, the process group ID of the calling process is returned.

#### Return\_value

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword in which the getpgid service returns a process group ID or, if it is not successful, a -1.

#### Return\_code

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword in which the getpgid service stores the return code. The getpgid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getpgid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EPERM</b>	The process whose process ID is equal to PID is not in the same session as the calling process.
<b>ESRCH</b>	There is no process with a process ID equal to PID.

#### Reason\_code

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword in which the getpgid service stores the reason code. The getpgid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Related services

- [“getsid \(BPX1GES, BPX4GES\) — Get the process group ID of the session leader” on page 317](#)
- [“getpgrp \(BPX1GPG, BPX4GPG\) — Get the process group ID” on page 297](#)
- [“getpid \(BPX1GPI, BPX4GPI\) — Get the process ID” on page 298](#)
- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ” on page 777](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GEP \(getpgid\) example” on page 1242](#).

# getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID

---

## Function

The getpgrp callable service gets the process group ID (PGID) of the calling process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GPG):</b>	31-bit
<b>AMODE (BPX4GPG):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GPG,(Group_ID)
```

## Parameters

### Group\_ID

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getpgrp service places the caller's process group ID.

**Usage notes**

If getpgrp fails, the process ends abnormally.

**Related services**

- “setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control ” on page 777
- “setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID ” on page 793

**Characteristics and restrictions**

None.

**Examples**

See “BPX1GPG (getpgrp) example” on page 1247.

## **getpid (BPX1GPI, BPX4GPI) — Get the process ID**

---

**Function**

The getpid callable service gets the process ID (PID) of the calling process.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GPI):</b>	31-bit
<b>AMODE (BPX4GPI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1GPI, (Process_ID)
```

**Parameters****Process\_ID**

Returned parameter

**Type:**

Integer

**Length:**

Fullword



The name of a fullword where the getpid service places the caller's process ID.

## Usage notes

1. If the getpid service fails, the process abends.
2. To optimize performance, see [Appendix I, “Optimizing performance using process- and thread-level information,” on page 1431](#).

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“getppid \(BPX1GPP, BPX4GPP\) — Get the parent process ID ” on page 299](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GPI \(getpid\) example” on page 1248](#).

# getppid (BPX1GPP, BPX4GPP) — Get the parent process ID

---

## Function

The getppid callable service gets the parent process ID (PPID) of the calling process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GPP):</b>	31-bit
<b>AMODE (BPX4GPP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GPP,(Return_value)
```

AMODE 64 callers use BPX4GPP.

## Parameters

### Return\_value

Returned parameter

### Type:

Integer

### Length:

Fullword

The name of a fullword in which the getppid service returns the parent process ID of the calling process.

## Usage notes

If the getppid service fails, the process ends abnormally.

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“getpid \(BPX1GPI, BPX4GPI\) — Get the process ID” on page 298](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GPP \(getppid\) example” on page 1248](#).

# getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process

---

## Function

The getpriority callable service gets the scheduling priority of a specific process or group of processes.

## Requirements

### Operation

### Environment

#### Authorization:

Supervisor or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1GPY):

31-bit

#### AMODE (BPX4GPY):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GPY,(Which,
              Who,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPY with the same parameters.

## Parameters

### Which

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates how the Who parameter is to be interpreted. This parameter can have one of the following values:

- PRIO\_PROCESS = Indicates that the Who parameter is to be interpreted as a process ID
- PRIO\_PGRP = Indicates that the Who parameter is to be interpreted as a process group ID
- PRIO\_USER = Indicates that the Who parameter is to be interpreted as a user ID

The PRIO\_ constants are defined in the BPXYCONS macro (see [“BPXYCONS — Constants used by services”](#) on page 1062).

### Who

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates the exact process ID, process group ID or User ID whose priority is to be obtained. The Which parameter indicates how this parameter is to be interpreted. A value of zero for this parameter specifies the current process, process group or User ID.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getpriority service returns the priority value of the specified process, or -1 if it is not successful.

Because the getpriority service can return the value -1 on successful completion, it is necessary to set the Return\_code parameter to 0 before a call to the getpriority service. If getpriority returns the value -1, the Return\_code parameter can be checked to see if the service was successful, or if an error occurred.

### Return\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the getpriority service stores the return code. The getpriority service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getpriority service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value of the Which parameter was not recognized; or the value of the Who parameter is not a valid process ID, process group ID or user ID.
<b>ESRCH</b>	No process could be located using the Which and Who parameter values that were specified.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getpriority service stores the reason code. The getpriority service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the supplied Who and Which values specify more than one process, the lowest priority value found among the specified processes is returned.
2. The setting of a process's priority value has an equivalent effect on a process's nice value, as they both represent the process's relative CPU priority. For example, setting the priority value of a process via the setpriority service to its maximum value (19) has the effect of increasing its nice value to its maximum value (2\*NICE\_ZERO)-1, and is reflected on the nice, getpriority and setpriority services. The NICE\_ZERO constant is defined in BPXYCONS. (See [“BPXYCONS — Constants used by services”](#) on page 1062.)

**Related services**

- [“setpriority \(BPX1SPY, BPX4SPY\) — Set the scheduling priority of a process”](#) on page 779
- [“nice \(BPX1NIC, BPX4NIC\) — Change the nice value of a process”](#) on page 497

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GPY \(getpriority\) example”](#) on page 1249.

## getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database

---

**Function**

The getpwent callable service gets information about a user. Each time you use the getpwent service, you get information about the next user entry in the user database.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GPE):</b>	31-bit
<b>AMODE (BPX4GPE):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GPE, (Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GPE with the same parameters.

## Parameters

### Return\_value

Returned parameter

#### Type:

Address

#### Length:

Fullword

The name of a fullword in which the getpwent service returns an address, or 0.

If no more user entries exist in the user database, or if an error is encountered, Return\_value is set to 0.

If an entry is found, Return\_value is set to the address of a data area that is mapped by the BPXYGIDN macro. The first area contains the fullword length of the user name, followed by the user name padded with blanks. For more information, see [“BPXYGIDN — Map data returned for getpwnam and getpwuid”](#) on page 1079.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getpwent service stores the return code, or 0. The getpwent service returns Return\_code only if Return\_value is 0. Return\_code is 0 when no more user entries exist in the database. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getpwent service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EMVSSAF2ERR</b>	The system authorization facility (SAF) or RACF Get GMAP service had an error.
<b>EMVSSAFEXTRERR</b>	The SAF or RACF RACROUTE EXTRACT service had an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getpwent service stores the reason code, or 0. The getpwent service returns Reason\_code only if Return\_value is 0. Reason\_code is 0 when no more user entries exist in the database. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see Reason codes in *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	12	Internal error during RACF processing
<b>8</b>	16	Unable to establish recovery
<b>8</b>	20	The current user is incompletely defined.

**Usage notes**

1. The getpwent service is intended to be used to search the user database sequentially. The first call to this service from a given task returns a pointer to the first user entry in the user database. Subsequent calls from the same task return a pointer to the next user entry found that is a defined OMVS user. When the end of the database is reached, a null pointer is returned (RV and RS are both set to 0).

A user entry is not returned for users that are not defined as OMVS users. This includes:

- Users that do not have a RACF OMVS segment defined
- Users with a RACF OMVS segment defined but with no UID defined
- Users whose DFLTGRP does not have a valid OMVS GID defined

If a user does not have a RACF OMVS segment but is given access to the OMVS DEFAULT user, no entry is returned on a getpwent call. This is because the user is not defined as an OMVS user in the user database.

2. The setpwent service can be used to reset this sequential search. The next getpwent service used from the same task after a call to the setpwent service returns a pointer to the first user entry. The next getpwent service used after an end-of-file indication (a null pointer) is returned also returns a pointer to the first user entry. The use of setpwent after end-of-file is therefore optional.
3. The return value points to data that may change or go away after the next getpwuid, getpwnam, or getpwent service request from that task. Each task manages its own storage separately. Move data to the program's storage if it is needed for future reference.
4. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.

**Related services**

- [“getpwnam \(BPX1GPN, BPX4GPN\) — Access the user database by user name ” on page 305](#)

- [“getpwuid \(BPX1GPU, BPX4GPU\) — Access the user database by user ID ” on page 308](#)
- [“setpwent \(BPX1SPE, BPX4SPE\) — Reset the user database ” on page 782](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GPE \(getpwent\) example” on page 1247.](#)

# getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name

---

## Function

The getpwnam callable service gets information about a user. You specify the user by user name.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GPN):</b>	31-bit
<b>AMODE (BPX4GPN):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GPN, (User_name_length,
               User_name,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GPN with the same parameters.

## Parameters

### User\_name\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of User\_name.

**User\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the User\_name\_length parameter

The name of a field of length User\_name\_length that contains the name of the user that the program wants information about. The name is specified in the Resource Access Control Facility (RACF) command that defines the user to the system.

**Return\_value**

Returned parameter

**Type:**

Address

**Length:**

Fullword

The name of a fullword in which the getpwnam service returns an address, or 0.

If no entry for the specified group name is found, Return\_value is set to 0.

If an entry is found, Return\_value is set to the address of the BPXYGIDN macro. See [“BPXYGIDN — Map data returned for getpwnam and getpwuid” on page 1079](#).

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getpwnam service stores the return code. The getpwnam service returns Return\_code only if Return\_value is 0. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getpwnam service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	User_name_length is incorrect; or the user name has an illegal first character (JRUserNameBad).
<b>EMVSSAF2ERR</b>	The system authorization facility (SAF) Get GMAP service had an error.
<b>EMVSSAFEXTRERR</b>	The SAF RACROUTE EXTRACT service had an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getpwnam service stores the reason code. The getpwnam service returns Reason\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



In the case of EMVSSAF2ERR or EMVSSAFEXTRERR, the reason code contains the RACF return and reason codes, respectively, in the two low-order bytes, as follows:

- For EMVSSAF2ERR, the reason code contains the return code and reason code from the RACF getGMAP service. For a detailed description of the return code and reason code values for the getGMAP service, see [getGMAP \(IRRSGM00\): Get GID-to-Group-Name mapping in z/OS Security Server RACF Callable Services](#).
- For EMVSSAFEXTRERR, the reason code contains the return code and reason code from the RACROUTE REQUEST=EXTRACT service. For a detailed description of the return code and reason code values for the RACROUTE service, see [RACROUTE REQUEST=AUTH \(standard form\) in z/OS Security Server RACROUTE Macro Reference](#).

RACF Get GMAP service return and reason code values include the following:

RACF return code	RACF reason code	Explanation
8	0	No profile found.
8	4	If the search is by GID: The GID is not defined. If the search is by group name: The current group is not defined.
8	8	The group name is not defined.
8	12	There was an internal error during RACF processing.
8	16	Recovery could not be established.
8	20	The current group is incompletely defined.

## Usage notes

1. If an entry for the specified User\_name is not found in the user database, an address of 0 is returned as the Return\_value parameter.
2. Return\_value points to data that may change or go away after the next getpwuid, getpwnam, or getpwent service request from that task. Each task manages its own storage separately. Move data to your own dynamic storage if you need it for future reference.
3. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX services.
4. If the BPX.UNIQUE.USER profile is defined, a call to BPX1GPN for a user ID that does not have an OMVS segment configured as part of its security profile causes BPX1GPN to assign OMVS segments to the user ID. See [Setting up default OMVS segments in z/OS UNIX System Services Planning](#).

## Related services

- [“getpwent \(BPX1GPE, BPX4GPE\) — Sequentially access the user database” on page 302](#)
- [“getpwuid \(BPX1GPU, BPX4GPU\) — Access the user database by user ID ” on page 308](#)
- [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name” on page 289](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GPN \(getpwnam\) example ” on page 1248](#).

## getpwuid (BPX1GPU, BPX4GPU) – Access the user database by user ID

---

### Function

The getpwuid callable service gets information about a user. You specify the user by user ID (UID).

### Requirements

#### Operation

#### Authorization:

#### Dispatchable unit mode:

#### Cross memory mode:

#### AMODE (BPX1GPU):

#### AMODE (BPX4GPU):

#### ASC mode:

#### Interrupt status:

#### Locks:

#### Control parameters:

#### Environment

Supervisor state or problem state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1GPU, (User_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GPU with the same parameter.

### Parameters

#### User\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the user ID of the user you want information about.

#### Return\_value

Returned parameter

#### Type:

Address

#### Length:

Fullword

The name of a fullword to which the getpwuid returns an address, or 0. If no entry for the specified user ID is found, Return\_value is set to 0. If an entry is found, Return\_value is set to the address

of the BPXYGIDN mapping macro. See [“BPXYGIDN — Map data returned for getpwnam and getpwuid” on page 1079](#).

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getpwuid service stores one of the following return codes:

Return_code	Explanation
<b>EMVSSAF2ERR</b>	The system authorization facility (SAF) or RACF Get GMAP or Get UMAP service had an error.
<b>EMVSSAFEXTRERR</b>	The SAF or RACF RACROUTE EXTRACT call had an error.

The getpwuid service returns Return\_code only if Return\_value is 0. For a list of return code values, see [Return codes \(errnos\) in z/OS UNIX System Services Messages and Codes](#)

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getpwuid service stores the reason code. The getpwuid service returns Return\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#). The reason code for EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	<b>4</b>	For a search by GID: the GID is not defined. For a search by group name: the current group is not defined.
<b>8</b>	<b>8</b>	The group name is not defined.
<b>8</b>	<b>12</b>	There was an internal error during RACF processing.
<b>8</b>	<b>16</b>	Recovery could not be established.
<b>8</b>	<b>20</b>	The current group is incompletely defined.

For a more detailed description of the RACF Get UMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	<b>4</b>	For a search by UID: the UID is not defined. For a search by user ID: The user is not defined.
<b>8</b>	<b>8</b>	The user ID is not defined.
<b>8</b>	<b>12</b>	An internal error occurred during RACF processing.
<b>8</b>	<b>16</b>	Unable to establish recovery.

<b>RACF return code</b>	<b>RACF reason code</b>	<b>Explanation</b>
8	20	The user is incompletely defined.

**Usage notes**

1. Return\_value points to data that can change or go away after the next getpwuid, getpwnam, or getpwent service request from the task. Each task manages its own storage separately. Move data to the program's dynamic storage if the program needs it for future reference.
2. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.
3. Most systems have multiple user IDs that are defined as UID=0. The SUPERUSER statement in BPXPRMxx specifies the user ID that the kernel will use when you need a user ID for UID(0), and the default for that user ID is BPXROOT. The installation might have specified a different user ID as the default user ID for the SUPERUSER statement. For more information about the SUPERUSER statement, see [BPXPRMxx \(z/OS UNIX System Services parameters\)](#) in *z/OS MVS Initialization and Tuning Reference*.
4. Performance degradation can occur if you use this service when the Virtual Lookaside Facility (VLF) is not active. For more information, see [Tuning performance](#) and [Establishing UNIX security](#) in *z/OS UNIX System Services Planning*.

**Related services**

- [“getpwent \(BPX1GPE, BPX4GPE\) — Sequentially access the user database” on page 302](#)
- [“getpwnam \(BPX1GPN, BPX4GPN\) — Access the user database by user name ” on page 305](#)
- [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name” on page 289](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GPU \(getpwuid\) example” on page 1249](#).

## getrandom (BPX1GRD, BPX4GRD) — Obtain a series of random bytes

---

**Function**

The **getrandom** service creates a new getrandom instance and returns random data that can be used for cryptographic purposes by filling up buffer address with up to buflen bytes.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PI2)</b>	31-bit
<b>AMODE (BPX4PI2)</b>	64-bit
<b>ASC mode</b>	Primary mode

<b>Operation</b>	<b>Environment</b>
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GRD (Flags,
              Buf_len,
              *Buf,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GRD with the same parameters.

Parameters

Flags

Supplied parameter.

Type

Integer

Length

Fullword

The name of a fullword field that indicates the Initialization options. This field is mapped by the BPXYGRD macro (see [“BPXYGRD — Map the response structure for getrandom”](#) on page 1080).

The values for this field are as follows:

Flag	Value	Description
<b>GRND_RANDOM</b>	'00001000'X	If this bit is set, then random bytes are drawn from the random source (such as the same source as the /dev/randomdevice). For z/OS UNIX, the same action applies to /dev/urandom. There is no limitation to the number of bytes returned. It is only limited by the buffer size.
<b>GRND_NONBLOCK</b>	'00000004'X	By default, if GRND_NONBLOCK flag is set, then getrandom() does not block. Blocking for getrandom() is not supported. Getrandom() will always return the data that is specified in the buf_len.

Buf\_len

Supplied parameter.

Type

Signed 32-bit integer.

Length

Word

Buf\_len refers to the user-specified buffer length.

buf

Parameter supplied and returned.

## getrandom (BPX1GRD, BPX4GRD)

### Type

Character string.

### Length

Specified by the `Buf_len` parameter.

`buf` refers to the user-specified buffer where the random data is returned.

### Return\_value

Returned parameter.

### Type

Integer

### Length

Fullword

A random data of the length `buf_len` or -1 if it is not successful.

### Return\_code

Returned parameter.

### Type

Integer

### Length

Fullword

The name of a fullword in which the **getrandom** service stores the return code. The **getrandom** service returns the `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The **getrandom** service can return one of the following values in the `Return_code` parameter:

Return_code	Explanation
<b>EFAULT</b>	The address referred to by <code>buf</code> is outside the accessible address space.
<b>EINTR</b>	The call was interrupted by a signal handler.
<b>EINVAL</b>	An invalid value was specified in <code>flags</code> .

### Reason\_code

Returned parameter.

### Type

Integer

### Length

Fullword

The name of a fullword in which the **getrandom** service stores the reason code. The **getrandom** service returns the `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_value`. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

`GRND_NONBLOCK` is not supported. Its behavior is the same as that of `GRND_RANDOM`.

## Related services

There are no related services for the **getrandom** service.

## Characteristics and restrictions

None.

**Examples**

“BPX1GRD (getrandom) example” on page 1250 and “BPX4GRD (getrandom) example” on page 1343.

## getrlimit (BPX1GRL, BPX4GRL) – Get resource limits

---

**Function**

The getrlimit callable service gets hard and soft resource limits for the calling process.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GRL):</b>	31-bit
<b>AMODE (BPX41GRL):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1GRL, (Resource,
               Rlimit,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GRL with the same parameters.

**Parameters****Resource**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains a value that indicates the resource for which to get the hard and soft limits. This parameter can have one of the following values:

Constant name	Description
<b>RLIMIT_MEMLIMIT</b>	Limit of 1-megabyte segments above the 2-gigabyte addressing range
<b>RLIMIT_CORE</b>	Limit size of core dump
<b>RLIMIT_CPU</b>	Limit CPU time per process

Constant name	Description
<b>RLIMIT_FSIZE</b>	Limit file size
<b>RLIMIT_NOFILE</b>	Limit number of open files
<b>RLIMIT_AS</b>	Limit address space size

The RLIMIT\_ constants are defined in the BPXYCONS macro. (See [“BPXYCONS — Constants used by services”](#) on page 1062.)

### Rlimit

Supplied parameter

#### Type

Structure

#### Length

The length of the Rlimit structure

The name of an Rlimit structure in which the hard (maximum) and soft (current) limit values for the resource that is identified by the resource parameter are to be placed. Macro BPXYRLIM defines the Rlimit structure. (See [“BPXYRLIM — Map the rlimit, rusage, and timeval structures”](#) on page 1142.) Each limit value contains two fullwords. For all resources except RLIMIT\_FSIZE and RLIMIT\_MEMLIMIT, the upper fullword for each limit value is set to zero before returning to the caller.

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the getrlimit service returns a value of zero if it is successful, and -1 if it is not successful.

### Return\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the getrlimit service stores the return code. The getrlimit service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getrlimit service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	An incorrect resource was specified. The following reason code can accompany the return code: JrInvalidResource.

### Reason\_code

Returned parameter

#### Type

Integer

#### Length

Fullword



The name of a fullword in which the getrlimit service stores the reason code. The getrlimit service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Related services

- [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ” on page 574](#)
- [“setrlimit \(BPX1SRL, BPX4SRL\) — Set resource limits ” on page 789](#)
- [“getrusage \(BPX1GRU, BPX4GRU\) — Get resource usage ” on page 315](#)

## Characteristics and restrictions

The current high-memory limit that is returned in RLIM\_CUR\_DW is the exact current memory limit. When the MEMLIMIT value is set by SMF, the maximum number of bytes that can be returned in RLIM\_MAX\_DW is X'FFFFFFFF00000000'. When the MEMLIMIT value is set by a z/OS UNIX service, the highest value that is supported is 16383 petabytes, or X'FFFC000000000000'. It is therefore possible for the current MEMLIMIT to be larger than the maximum MEMLIMIT supported by z/OS UNIX System Services.

## Examples

See [“BPX1GRL \(getrlimit\) example” on page 1250](#).

# getrusage (BPX1GRU, BPX4GRU) — Get resource usage

## Function

The getrusage callable service gets information about resources that are used by the calling process or its terminated and waited-for child processes.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GRU):</b>	31-bit
<b>AMODE (BPX4GRU):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GRU,(Who,
               Rusage,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GRU with the same parameters.

## Parameters

### Who

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates for whom to get the resource usage. This parameter can have one of the following values:

#### **RUSAGE\_SELF**

Rusage for current process

#### **RUSAGE\_CHILDREN**

Rusage for terminated children

The RUSAGE\_ constants are defined in the BPXYCONS macro (see [“BPXYCONS — Constants used by services”](#) on page 1062).

### Rusage

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of an rusage structure that is to contain the values for resource usage. Macro BPXYRLIM defines the rusage structure (see [“BPXYRLIM — Map the rlimit, rusage, and timeval structures”](#) on page 1142).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getrusage service returns a value of zero if it is successful, and -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getrusage service stores the return code. The getrusage service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The getrusage service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	An incorrect Who value was specified. The following reason code can accompany the return code: JrInvalidWho.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getusage service stores the reason code. The getusage service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

Resource information returned for multithreaded processes may be inaccurate.

**Related services**

- “[prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes](#)” on page 574
- “[getrlimit \(BPX1GRL, BPX4GRL\) — Get resource limits](#)” on page 313
- “[setrlimit \(BPX1SRL, BPX4SRL\) — Set resource limits](#)” on page 789

**Characteristics and restrictions**

None.

**Examples**

See “[BPX1GRU \(getusage\) example](#)” on page 1250.

## getsid (BPX1GES, BPX4GES) — Get the process group ID of the session leader

---

**Function**

The getsid callable service gets the process group ID of the session leader of the process whose process ID is equal to the input process ID. If the input process ID is 0, the service returns the process group ID of the session leader of the calling process.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GES):</b>	31-bit
<b>AMODE (BPX4GES):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GES,(PID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GES with the same parameter.

## Parameters

### PID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the process ID that identifies the process whose session leader's process group ID should be obtained. If PID is 0, the process group ID of the calling process's session leader is returned.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getsid service returns a process group ID, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getsid service stores the return code. The getsid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getsid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EPERM</b>	The process whose process ID is equal to PID is not in the same session as the calling process.
<b>ESRCH</b>	There is no process with a process ID equal to PID.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getsid service stores the reason code. The getsid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Related services

- [“setuid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ” on page 793](#)
- [“getpgid \(BPX1GEP, BPX4GEP\) — Get the process group ID” on page 295](#)
- [“getpgrp \(BPX1GPG, BPX4GPG\) — Get the process group ID” on page 297](#)
- [“getpid \(BPX1GPI, BPX4GPI\) — Get the process ID” on page 298](#)
- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ” on page 777](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GES \(getsid\) example” on page 1242](#).

# getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer

---

## Function

The getsockname or getpeername callable service obtains the name of a socket or the name of a peer connected to a socket.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GNM):</b>	31-bit task or SRB mode
<b>AMODE (BPX4GNM):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

AMODE 64 callers use BPX4GNM with the same parameters.

## Format

The syntax format is as follows:

```
CALL BPX1GNM, (Socket_descriptor,
               Operation,
               Sockaddr_length,
               Sockaddr,
```

```
Return_value,  
Return_code,  
Reason_code)
```

## Parameters

### Socket\_descriptor

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the socket file descriptor for which the service is to be performed.

### Operation

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the operation option. The value of this field determines whether the service to be performed is getsockname or getpeername. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for valid Operation values.

### Sockaddr\_length

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length of Sockaddr. On return, this field specifies the size required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information contained in Sockaddr is truncated to the length supplied on input. The size of this field must be less than 4096 bytes (4KB) in length. The size of the buffer that is specified must be the maximum length that the sockaddr could be on output.

### Sockaddr

Supplied and returned parameter

**Type:**

Character

**Length:**

Length specified by Sockaddr\_length.

The name of a field in which the socket name or peer name is to be returned. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for valid Operation values.

### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getsockname or getpeername service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getsockname or getpeername service stores the return code. The getsockname or getpeername service returns Return\_code only if Return\_value is -1. For a list of return code values, see *Return codes (errno) in z/OS UNIX System Services Messages and Codes*. The getsockname or getpeername service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINVAL</b>	The length that is specified by the sockaddr_length operand is too small to allow the name to be returned. The following reason code can accompany the return code: JRSocketCallParmError.
<b>ENOBUFS</b>	Unable to obtain a buffer. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JROutOfSocketCells.
<b>ENOTCONN</b>	getpeername() was specified and the socket is not connected.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getsockname or getpeername service stores the reason code. The getsockname or getpeername service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GNM \(getpeername or getsockname\) example” on page 1247](#).

## getsockopt or setsockopt (BPX1OPT, BPX4OPT) – Get or set options associated with a socket

---

### Function

The getsockopt or setsockopt callable service gets or sets options that are associated with a socket.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1OPT):</b>	31-bit task or SRB mode
<b>AMODE (BPX4OPT):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1OPT,(Socket_descriptor,
              Operation,
              Level,
              Option_name,
              Option_data_length,
              Option_data,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4OPT with the same parameters.

### Parameters

#### Socket\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the service is to be performed.

#### Operation

Supplied parameter

##### Type:

Integer



**Length:**

Fullword

The name of a field that contains the operation option. The value of this field determines whether the service to be performed is getsockopt, setsockopt, or setibmssockopt. See [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 for valid Operation values.

**Level**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the level for which the option is set or being set.

**Option\_name**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field in which the value of the option name is provided. See [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 for valid Option\_name values.

**Option\_data\_length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length of Option\_data. On return from getsockopt, this field contains the size of the data that was returned in Option\_data. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer specified should be the maximum length that the option\_data could be on output.

**Option\_data**

Supplied and returned parameter

**Type:**

Character

**Length:**

Length specified by Option\_data\_length.

The name of a field that contains the data that is associated with or is to be associated with the socket. On return from getsockopt, this field contains the data that is associated with the socket. For setsockopt, this field provides the data that is to be associated with the socket.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getsockopt or setsockopt service returns one of the following:

- 0 if the request is successful.

- -1 if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getsockopt or setsockopt service stores the return code. The getsockopt or setsockopt service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getsockopt or setsockopt service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EDOM</b>	An argument that is too large was supplied on the call.
<b>EINVAL</b>	An incorrect argument was supplied on the call. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRLevelNotSupp.
<b>ENOBUFS</b>	A buffer could not be obtained. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENPROTOOPT</b>	An option_name that was specified for getsockopt is not supported. An incorrect value was specified on the Level parameter. SOL_SOCKET must be specified. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRLevelNotSupp, JRInvalOpOpt, JROptNotSupp.
<b>ENOSYS</b>	For AF_UNIX, setsockopt was specified; it is not supported. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSetNotSupp.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRMustBeSocket.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getsockopt or setsockopt service stores the reason code. The getsockopt or setsockopt service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The socket descriptor must refer to an open socket.
2. For AF\_UNIX sockets, the getsockopt() service supports the following option\_names only: so\_acceptconn, so\_type, and so\_secinfo.
3. The level of support for this service depends on the particular socket stack you have installed. Some options might not be defined by the BPXYSOCK macro. Refer to the documentation for the product you

are using to determine the socket options it supports. For example, see *z/OS Communications Server: IPv6 Network and Appl Design Guide* for the z/OS Communications Server socket stack.

4. For more information about programming considerations for SRB mode, see [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041.

## Characteristics and restrictions

None.

## Examples

See [“BPX1OPT \(getsockopt or setsockopt\) example”](#) on page 1268.

# **\_\_getthent (BPX1GTH, BPX4GTH) – Get thread data**

---

## Function

The \_\_getthent callable service obtains data that describes the status of a process and its threads. This data includes, but is not limited to, running time, reasons for waiting, syscalls made, files open, and signal information. The caller can access one process on each request, with from none to all of its threads.

You can invoke this service in several ways:

- For the first accessible (by SAF standards) process (the lowest relative process in the system).
- For a specific process, if the process ID is known and the process is accessible.
- For a specific thread within a specific accessible process, if both IDs are known.
- For the next accessible process or thread after one is returned.
- For a specific address space ID or user ID.

## Requirements

Operation	Environment
<b>Authorization</b>	Problem program or supervisor state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1GTH)</b>	31-bit
<b>AMODE (BPX4GTH)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	No latches should be held
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GTH, (Input_length,
               Input_address,
               Output_length,
               Output_address,
               Return_value,
```

```
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4GTH with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Input\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of the fullword that contains the value PGTHA#LEN.

### Input\_address

Supplied parameter

#### Type

Address

#### Length

Fullword (doubleword)

The name of the fullword (doubleword) that contains the address of an area that is mapped by PGTHA; see [“BPXYPGTH — Map the \\_\\_getthent input/output structure”](#) on page 1119. The input area must be initialized to hex zeros and then the requested options must be set in the PGTHA section.

### Output\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of the fullword that contains the length of the output buffer. Some requests might be satisfied by the minimum buffer size of 128 bytes while a request for all options of a process with maximum resources might exceed half a million bytes.

### Output\_address

Supplied parameter

#### Type

Address

#### Length

Fullword (doubleword)

The name of the fullword (doubleword) that contains the address of an area that is mapped by PGTHB and other PGTH sections; see [“BPXYPGTH — Map the \\_\\_getthent input/output structure”](#) on page 1119.

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the \_\_getthent service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the \_\_getthent service stores the return code. The \_\_getthent service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_getthent service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	The <b>input_addr</b> (to <b>input_length</b> ) or the <b>output_addr</b> (to <b>output_length</b> ) contains the address of storage that the caller is not authorized to access (JrBadInputError or JrBadOutputError).
<b>EINVAL</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• The input area (PGTHA) contains a value that is not valid (JrPidBad, JrBadOptions)</li> <li>• The <b>input_address</b> is zero (JrBadInputBuffAddr)</li> <li>• The <b>output_address</b> is zero (JrBadOutputBuffAddr)</li> <li>• The <b>input_length</b> is incorrect (JrInvParmLength)</li> <li>• The <b>output_length</b> is too small (JrBuffTooSmall)</li> </ul>
<b>EAGAIN</b>	PGTHAPID is undergoing changes, and the z/OS UNIX control blocks are not properly connected (JrBlocksInFlux).
<b>EACCES</b>	PGTHAPID is not accessible to the caller (JrInaccessible).
<b>ESRCH</b>	PGTHAPID was not found (JrPIDNotFound); or PGTHATHID was not found (JrThreadNotFound).
<b>ENOBUFS</b>	The __getthent service could not obtain a local work area (JrNoBufStorage).

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_getthent service stores the reason code. The \_\_getthent service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes for \_\_getthent**

1. When PGTHATHREAD (PGTHJ) and PGTHATHDSIGMASK are specified the thread's signal-blocking mask is returned in the PGTHJSIGMASK field and the PGTHJSIGMASK field is mapped over the last two entries of the PGTHJPREVSC, which limits the system call history to three entries.
2. Typically, the \_getthent service uses process-level security when determining what data can be seen by the invoker. To use thread-level identity instead, specify the `_BPXK_GPSENT_SECURITY=THREAD` environment variable. For more information, see [Commonly used environment variables](#) in *z/OS UNIX System Services Planning*.

3. The system extracts fields **PgthJTTime** and **PgthJLoginName** from the TCB and OTCB of the target process by scheduling an SRB to run in that process's address space. If the address space is terminating, the SRB might not run, and these fields are returned as zero.

If this information is not required, you can set option PgthAThreadFast. This action can improve performance on a busy system because the SRB will not be scheduled. An indication of the thread being in an MVSWAIT is also extracted from the TCB. If the thread is in an MVS WAIT, then PgthJStatus is set to Y. When the SRB is not scheduled, then this field is not seen as Y. If the thread is not in any other kernel wait, then PgthJStatus2 is set to R, which indicates running or non-kernel wait.

4. Typically a user starts with PGTHAPID=PGTH#FIRST, processes the data, and sets PgthAContinue=PgthBContinue to continue with the next thread or the next process, until a Return\_value of -1 is reached.
5. The setting of PgthBContinue steps the caller to the next process or thread. If this action is not desirable, do not use PgthAContinue.
6. The Output\_length required varies with the PGTHAFLAGS selected and the characteristics of the process and its threads. Most processes should fit in 4000 bytes. An example of when a bigger buffer is needed is if PgthAFileData is selected and the process has 65 000 files opened. In this case,  $\frac{3}{4}$  of a million bytes is needed. If PgthAProcess is the only input option that is selected, then the buffer must be at least 176 bytes to get back any data.
7. EINVAL (JrBuffTooSmall) can also indicate that there is insufficient room for at least one PgthJ area. This situation could happen even with a buffer in excess of 4000 bytes.
8. The PgthJWTime field indicates how long the thread has been in most waits that are internal to z/OS UNIX. It is meaningful only if it is nonzero.
9. The high-memory values for the process that are returned in PgthCMemUsage and PgthCMemPages are described in bytes. Each value can be displayed as a 4-byte number or as a 3-byte value followed by a qualifier (for example, 50M for fifty megabytes). High-memory values are described with the truncated value of the exact system state; the real values might be slightly higher. The maximum number of bytes that can be returned in PgthCMemUsage and PgthCMemPages is 16383 petabytes.
10. Flag PGTHCRESPAWN indicates that the process was started with the respawn attribute. See [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process”](#) on page 869 for more information about the respawn attribute.
11. Flag PGTHCBLOCKING indicates a shutdown-blocking process. Flag PGTHCPERM indicates a permanent process.
12. When the command or argument data is not terminated by a null character, the command or argument is terminated with a null character by the \_\_getthent service.
13. If PgthACommand is ON and PgthACommandLong is OFF, the maximum length that is returned in the PGTHF area is 1024. If PgthACommand is OFF and PgthACommndLong is ON, the length that is returned in the PGTHF area can be greater than 1024. If both bits are on, the new setting, PgthACommndLong, is honored.
14. Together, the PgthE output area (for path) and the PgthF output area (for the command and arguments) can take up to 2048 bytes space. When PgthACommand is specified, 1024 bytes are reserved for the PgthE section (path name) and 1024 bytes are reserved for the PgthF section (the command and arguments). Up to only 1024 bytes of data are returned for the PgthF section.  
  
When PgthACommandLong is specified, it indicates that the PgthF section can be greater than 1024 bytes. If path is requested (PgthAPath), then the PgthE section is filled in and the remainder of the 2048 bytes can be used for the PgthF section. (The PgthF section length will be 2048 bytes less the length of the PgthE section returned). If path is not requested, then the PgthF section can be up to 2048 bytes.  
  
If PgthACommand and PgthACommandLong are both specified, PgthACommandLong is accepted.
15. PgthaFilePath must be specified with PgthaFileData. If it is not, then PgthaFilePath is ignored. If PgthaFilePath is specified, a significant amount of data might be returned if the process has many open files. Information is not returned if more than 16 MB is required or if the input buffer is too

small. The path names that are returned are full, relative, or partial and is intended for diagnostic use. If a partial path name is returned, the PgthHPathTrunc bit is set.

16. Current fields PgthCUserTime, PgthCSysTime are time values in 1/100th of a second. Current fields PgthJTTime, and PgthJWTime are time values in milliseconds. PgthCUserTimeMics, PgthCSysTimeMics, PgthJTTimeMics, and PgthJWTimeMics are those same time values, in microseconds.
17. You cannot use the length of PGTHB to get to the PGTHx areas. For example, when using PGTHC you must define it as address of PGTHB + PGTHBOFFC. You cannot base it off the length of the PGTHB structure. For a full list of the PGTHBOFF offsets, see [“BPXYPGTH — Map the \\_\\_getthent input/output structure” on page 1119](#).

## Related services

- [“w\\_getpsent \(BPX1GPS\) — Get process data ” on page 999](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GTH \(\\_\\_getthent\) example” on page 1251](#).

# getuid (BPX1GUI, BPX4GUI) — Get the real user ID

## Function

The getuid callable service gets the real user ID (UID) of the calling process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GUI):</b>	31-bit
<b>AMODE (BPX4GUI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax is as follows:

```
CALL BPX1GUI, (User_ID)
```

AMODE 64 callers use BPX4GUI.

## Parameters

### User\_ID

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword to which the getuid service returns the real user ID of the calling process.

## Usage notes

If the getuid service fails, the process ends abnormally.

## Related services

- [“getuid \(BPX1GEU, BPX4GEU\) — Get the effective user ID” on page 264](#)
- [“setuid \(BPX1SEU, BPX4SEU\) — Set the effective user ID” on page 760](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GUI \(getuid\) example” on page 1252](#).

# getwd (BPX1GWD, BPX4GWD) — Get the path name of the working directory

---

## Function

The getwd callable service gets the path name of the working directory.

## Requirements

### Operation

### Environment

#### Authorization:

Supervisor state or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1GWD):

31-bit

#### AMODE (BPX4GWD):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.



## Format

The syntax format is as follows:

```
CALL BPX1GWD,(Buffer_length,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GWD with the same parameters.

## Parameters

### Buffer\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the buffer to which the getwd service returns the pathname of the directory. Buffer\_length must be large enough to accommodate the actual length of the pathname plus one (for the terminating null). Length of zero has special meaning; see the usage notes.

### Buffer

Parameter supplied and returned

#### Type:

Character string

#### Character set:

No restrictions

#### Length:

Specified by the Buffer\_length parameter

The name of the buffer that will hold the pathname of the working directory.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getwd service returns the length of the path name that is in the buffer, if the request is successful, or -1, if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the getwd service stores the return code. The getwd service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The getwd service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The process did not have permission to read or search a component of the working directory's pathname.
<b>EINVAL</b>	The Buffer_length specified was not valid. The following reason code can accompany the return code: JRBufLenInvalid.
<b>EIO</b>	An input/output error occurred.
<b>ENOENT</b>	A component of a pathname does not exist. This is returned if a component of the working directory pathname was deleted.
<b>ERANGE</b>	The specified Buffer_length is less than the length of the pathname of the working directory. The specified Buffer_length is zero, and the length of the pathname of the working directory is larger than PATH_MAX bytes.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the getwd service stores the reason code. The getwd service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. PATH\_MAX plus 1 for the terminating null character is a reasonable size for the Buffer.
2. If a Buffer\_length of zero is passed to this service, the generated null terminated pathname is stored in the named buffer up to a maximum of PATH\_MAX + 1 bytes. Buffer is assumed to be of sufficient size to contain the pathname derived by the getwd service. If the generated pathname is larger than PATH\_MAX bytes, the return value is -1 and Return\_code is ERANGE.

**Related services**

- [“getcwd \(BPX1GCW, BPX4GCW\) — Get the path name of the working directory”](#) on page 260

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1GWD \(getwd\) example”](#) on page 1252.

## **givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program**

---

**Function**

The givesocket callable service makes a specified socket available to a takesocket call to be issued by another program. Any socket can be given. Typically, givesocket() is used by a main program that obtains sockets by means of accept() and gives them to application programs that handle one socket at a time.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1GIV)</b>	31-bit
<b>AMODE (BPX4GIV)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1GIV,(Socket_descriptor,
              Clientid,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GIV with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the socket file descriptor for which the givesocket is to be done.

### Clientid

Supplied and returned parameter.

#### Type

Structure.

#### Length

Length of BPXYCID.

The name of a structure that contains Clientid information identifying the subsidiary program to which the socket is to be given. This information is typically obtained with the getclientid service issued by the subsidiary and passed to the server. Clientid input may be as follows:

#### CIdDomain

The domain of the socket being given. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the values defined for this field.

#### CIdName

One of the following:

- Blanks, which allow any program to take the socket using the takesocket service.

- The subsidiary program's address space name, left-aligned, and padded with blanks.
- A fullword of binary zeros followed by the subsidiary program's process ID.

**CidTask**

Used only if an address space name was supplied in the **CidName** field. One of the following:

- Blanks, which allow any subtask in the address space to take the socket.
- The subsidiary program's subtask identifier.

**CidReserved**

One of the following:

- The **CidType** field of the CidReserved area set to Cid#Close. This results in the givesocket service doing a close of the input socket and returning a unique socket token in the **CidSockToken** field of the CidReserved area.
- The **CidType** field of the CidReserved area set to Cid#Select. This indicates that the application intends to block on select() for exceptions, waiting for the takesocket call to occur before closing the socket. It allows a select call to return exception status even if that call is made after the takesocket call, as long as the socket was not closed. It also results in the connection being severed if the giver closes the socket before it has been taken.
- The **CidType** field of the CidReserved area set to zeros. The program will not call select() to coordinate with the taker of the socket. Either the socket is not going to be closed or the giver and taker have some other method of coordination for the giver to know when the taker has called takesocket().

**Note:** If select() for exception is called before takesocket(), the select will return when takesocket() is called. If select() is called after takesocket(), it will hang. Use **CidType** of Cid#Select if select() will be called by the giver.

If the given socket is closed before the takesocket() is issued, it is possible for that socket descriptor number to be reused in the giver's process. A sequence of accept(), givesocket(), and close() calls issued several times before any takesocket() calls can result in several sockets with the same descriptor number waiting to be taken. In this case, the oldest given socket is taken and first in first out order is used.

**Note:** If select() is called when there are several given sockets with the same descriptor number waiting to be taken, then select() will operate on the current active socket for that descriptor. It effectively waits for the last (newest) given socket to be taken and the order becomes last in first out.

The Clientid is a returned parameter only if the **CidType** field in the CidReserved area is set to Cid#Close. A unique token for the given socket is then returned in the **CidToken** field of the CidReserved area. This token, instead of the socket descriptor, is to be passed to the subsidiary program to be used on the takesocket service. The token must be used, rather than the socket descriptor, because the socket being given will be closed and the socket descriptor might be reused. See [“BPXYCID — Map the returning structure for getclientid\(\)” on page 1061](#) for more information about the format of this field.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the givesocket service returns one of the following:

- 0 if successful.
- -1 if the request is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the givesocket service stores the return code. The givesocket service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The givesocket service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The Socket_descriptor is not valid, or the socket has already been given.
<b>EFAULT</b>	Using the Clientid parameter as specified would result in an attempt to access storage that is outside the caller's address space.
<b>EINVAL</b>	The Clientid parameter does not specify a valid client identifier; or the CIdDomain in the Clientid parameter does not match the actual domain of the input Socket_descriptor.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the givesocket service stores the reason code. The givesocket service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The program that does the givesocket will always be able to do a takesocket for the given socket, even if its identity does not match that of the Clientid input of the givesocket.
2. The Clientid output of the getclientid service (issued by the subsidiary program and passed to the server) is intended to be used as the input Clientid of the givesocket service. If you use a FunctionCode of 2 on the getclientid service to obtain Clientid information that will then be used as the Clientid input of the givesocket service, you will ensure the best performance of the givesocket service, and the most secure identification of the validity of the taker.
3. Setting the **CIdType** field of the CIdReserved area in the Clientid structure to CId#Close improves performance, by allowing the givesocket service to automatically close the socket, rather than requiring the application to do a select and a close.
4. If the given socket is not closed, it can still be used, even after the takesocket() has been done. The socket can be shared between the giver and taker in the same way that an inherited socket can be shared between parent and child after a fork() has been issued.

**Related services**

- [“getclientid \(BPX1GCL, BPX4GCL\) — Obtain the calling program's identifier” on page 258](#)
- [“takesocket \(BPX1TAK, BPX4TAK\) — Acquire a socket from another program ” on page 916](#)

**Characteristics and restrictions**

None.

## grantpt (BPX1GPT, BPX4GPT) – Grant access to the subsidiary pseudoterminal

---

### Function

The grantpt callable service changes the mode and ownership of the subsidiary pseudoterminal device that is identified by the file descriptor. The file descriptor must be the file descriptor of the corresponding manager pseudoterminal. The user ID of the subsidiary pseudoterminal is set to the real UID of the calling process. The group ID is set to the group ID that is associated with the group name that was specified by the installation in the TTYGROUP initialization parameter. The permission mode of the subsidiary pseudoterminal is set to be readable and writable by the owner, and writable by the group.

You can provide secure connections either by using grantpt and unlockpt, or by issuing the first open against the subsidiary pseudoterminal from the user ID or process that opened the manager pseudoterminal.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1GPT)</b>	31-bit
<b>AMODE (BPX4GPT)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1GPT,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPT with the same parameter.

### Parameters

#### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor for the terminal.

#### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the grantpt service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the grantpt service stores the return code. The grantpt service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The grantpt service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The subsidiary pseudoterminal was opened before grantpt(), or a grantpt() was already issued. In either case, subsidiary pseudoterminal permissions and ownership have already been updated. If you use grantpt() to change subsidiary pseudoterminal permissions, you must issue grantpt() between the manager open and the first pseudoterminal open, and grantpt() can only be issued once.
<b>EBADF</b>	The File_descriptor parameter does not specify a valid open file descriptor.
<b>EINVAL</b>	The file descriptor is not associated with a manager pseudoterminal device.
<b>ENOENT</b>	The subsidiary pseudoterminal device was not found during lookup.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the grantpt service stores the reason code. The grantpt service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“unlockpt \(BPX1UPT, BPX4UPT\) — Unlock a pseudoterminal manager and subsidiary pair ” on page 973](#)

**Characteristics and restrictions**

None.

## inotify\_add\_watch (BPX1IAW, BPX4IAW) – Add or modify a watch to an initialized inotify instance

---

### Function

The **inotify\_add\_watch** service manipulates the "watch list" associated with an inotify instance that is returned by the **inotify\_init** call. Each watched item in the watch list specifies the path name of a file or directory. Each watched item also specifies a set of events to be monitored by the kernel for the file referred to by that path name. The **inotify\_init** call either creates a new watch item or modifies an existing watch. Each watch has a unique watch descriptor, which is an integer that is returned by the call to **inotify\_add\_watch** when the watch item is created.

For the macro, see [“BPXYINFY— Map inotify\\_init syscall parameters” on page 1080](#).

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1IAW)</b>	31-bit
<b>AMODE (BPX4IAW)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1IAW(File_Descriptor,
             Path_Length,
             Path_Name,
             Mask,
             Return_value,
             Return_code,
             Reason_code)
```

AMODE 64 callers use BPX4IAW with the same parameters.

### Parameters

#### File\_Descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The inotify file descriptor is an integer that represents an instance of the inotify that is returned by the **inotify\_init** call.

#### Path\_Name\_Length

Supplied parameter.



**Type**

Integer

**Length**

Fullword

An integer that represents the length of a path name.

**Path\_Name**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

A string of characters that represents the path name of a file or directory.

**mask**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

z/OS UNIX supported the masks in the following table.

Mask	Explanation
<b>IN_ACCESS</b>	The file was accessed, for example, by the read or execve services. This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_ATTRIB</b>	The metadata (such as permissions, timestamps, extended attributes, link count, and user or group ID) was changed. This change occurs for both the directory itself and for objects inside the directory.
<b>IN_CLOSE_WRITE</b>	The file that was opened for writing was closed. This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_CLOSE_NO_WRITE</b>	A file or directory that was not opened for writing was closed. This change occurs for both the directory itself and the objects inside the directory.
<b>IN_CREATE</b>	A file or directory that is created in the watched directory (for example, open() O_CREAT, mkdir(), link(), symlink(), bind() on a UNIX domain socket). This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_DELETE</b>	File or directory that is deleted from the watched directory. This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_DELETE_SELF</b>	Watched file or directory was deleted. This event also occurs if an object is moved to another file system because <b>mv</b> in effect copies the file to the other file system and then deletes it from the original file system. An IN_IGNORED event is also generated for the watch descriptor.
<b>IN_MODIFY</b>	File was modified (for example, by write or truncate). This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_MOVE_SELF</b>	Watched file or directory was moved.

Mask	Explanation
<b>IN_MOVED_FROM</b>	Generated for the directory that contains the old file name when a file is renamed. This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_MOVED_TO</b>	Generated for the directory that contains the new file name when a file is renamed. This change occurs only for objects inside the directory, not for the directory itself.
<b>IN_OPEN</b>	File or directory was opened. This change occurs for both the directory itself and objects inside the directory.

#### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

On success, returns a watch descriptor (a nonnegative integer) or -1 if it is not successful.

#### Return\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

On error, when the return\_value is -1, the following return codes are possible.

Return_code	Explanation
<b>EACCES</b>	Read access to the specified file is not permitted.
<b>EBADF</b>	The file descriptor is not valid.
<b>EEXIST</b>	<i>mask</i> contains IN_MASK_CREATE and <i>pathname</i> refers to a file that is already being watched by the same <i>file descriptor</i> .
<b>EFAULT</b>	<i>pathname</i> points outside of the process's accessible address space.
<b>EINVAL</b>	The event <i>mask</i> contains no valid events; or <i>mask</i> contains both IN_MASK_ADD and IN_MASK_CREATE; or file descriptor is not an <b>inotify</b> file descriptor.
<b>ENAMETOOLONG</b>	<i>pathname</i> too long.
<b>ENOENT</b>	A directory component in <i>pathname</i> does not exist or has a dangling symbolic link.
<b>ENOMEM</b>	Not enough kernel memory was available. was available.
<b>ENOSPC</b>	The user ulimit on the total number of inotify watches was reached or the kernel failed to allocate a needed resource.
<b>ENOTDIR</b>	<i>mask</i> contains IN_ONLYDIR and <i>pathname</i> is not a directory.

#### Reason\_code

Returned parameter.

#### Type

Integer

**Length**

Fullword

A fullword in which the **inotify\_add\_watch** service stores the reason code. The Reason\_code is set only if Return\_value is -1. Reason\_code further qualifies the Return\_value.

**Usage notes**

1. The **inotify\_add\_watch** service manipulates a list of watches that are associated with an inotify instance. Each watched item in the watch list specifies the path name of a file or directory, along with a set of events that are monitored for the file referred to by that path name.
  - It can either set a new watch item or modify an existing watch. Each watch has a unique watch descriptor when it is created.
  - When events occur for monitored files and directories, those events can be read by the application into structured data described in [“Reading events from an inotify file descriptor” on page 343](#).

**Related services**

- [“inotify\\_init \(BPX1INI, BPX4INI\) — Initialize an inotify instance” on page 341](#)
- [“inotify\\_rm\\_watch \(BPX1IRW, BPX4IRW\) — Remove existing watches from inotify instances” on page 344](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

**Characteristics and restrictions**

None.

**Examples**

[“BPX1IAW \(inotify\\_add\\_watch\) example” on page 1252](#) and [“BPX4IAW \(inotify\\_add\\_watch\) example” on page 1346](#).

## **inotify\_init (BPX1INI, BPX4INI) — Initialize an inotify instance**

---

**Function**

The **inotify\_init** service monitors file system events by creating a new inotify instance and returning a file descriptor that refers to the new instance. The returned instance can then be used to watch files or directories by using the **inotify\_add\_watch** service.

For the macro, see [“BPXYINFY— Map inotify\\_init syscall parameters” on page 1080](#).

**Requirements**

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1INI)</b>	31-bit
<b>AMODE (BPX4INI)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts

<b>Operation</b>	<b>Environment</b>
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

```
CALL BPX1INI,(Options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4INI with the same parameters.

**Parameters**

**Options**  
IN\_NONBLOCK  
Sets the nonblock file status flag on the file description referred to by the new file descriptor.  
IN\_CLOEXEC  
Sets the close-on-exec flag on the new file descriptor.

**Return\_value**  
Returned parameter.  
**Type**  
Integer  
**Length**  
Fullword  
On success, returns a new file descriptor or -1 if it is not successful.

**Return\_code**  
Returned parameter.  
**Type**  
Integer  
**Length**  
Fullword  
When the return\_value is -1, the following return codes are possible.

Return_code	Explanation
EINVAL	An invalid value was specified in <i>flags</i> .
EMFILE	The user limit on the total number of <b>inotify</b> instances was reached.
EMFILE	The per-process limit on the number of open file descriptors was reached.
ENFILE	The system-wide limit on the total number of open files was reached.
ENOMEM	Not enough kernel memory is available.

**Reason\_code**  
Returned parameter.  
**Type**  
Integer  
**Length**  
Fullword

A fullword in which the **inotify\_init** service stores the reason code. The Reason\_code is set only if Return\_value is -1. Reason\_code further qualifies the Return\_value.

## Reading events from an inotify file descriptor

To determine what events have occurred, an application reads from the inotify file descriptor. If no events have occurred up to this point, then assuming a blocking file descriptor, a read will block until at least one event occurs.

A list of events is chained together with each event defined in the following structure. However, a successful read of the inotify descriptor returns a buffer that contains one or more of the INOTIFY\_EVENT structures.

0	CHARACTER	4	INNO_EventID	Eyecatcher 'INEV'
4	UNSIGNED	4	INNO_EventLen	Event length
8	STRUCTURE		INOTIFY_EVENT	inotify event structure
8	SIGNED	4	INOTIFY_wd	Watch descriptor
12	BIT	32	INOTIFY_Events	Mask describing event
16	UNSIGNED	4	INOTIFY_Cookie	Unique cookie
20	UNSIGNED	4	INOTIFY_Length	Size of name field
24	CHARACTER	*	INOTIFY_Name	Optional name

- The **INOTIFY\_wd** field identifies the watch for which this event occurs. It is one of the watch descriptors that were returned by a call to BPX1IAW.
- The **INOTIFY\_Events** field contains bits that describe the inotify events that occurred.
- The **INOTIFY\_Cookie** field is a unique integer that connects related events. The application connects the resulting pair of IN\_MOVED\_FROM and IN\_MOVED\_TO events. Use it for rename events only.
- The **INOTIFY\_Length** field counts all of the bytes in name. The length of each **inotify\_event** structure is the dimension of **notify\_event** struct and **INOTIFY\_Length**.
- The **INOTIFY\_Name** field is displayed when an event is returned for a file that is inside a watched directory. It identifies the file name in the watched directory.

The following inotify events are returned by a read() call.

Event	Description
IN_ACCESS (+)	File was accessed.
IN_ATTRIB (*)	Metadata was changed for permissions.
IN_CLOSE_WRITE (+)	File that was opened for writing was closed.
IN_CLOSE_NOWRITE (*)	File or directory that was not opened for writing was closed.
IN_CREATE (+)	File or directory was created in the watched directory.
IN_DELETE (+)	File or directory was deleted from the watched directory.
IN_DELETE_SELF (+)	The watched file or directory was itself deleted.
IN_MODIFY (+)	The file was modified.
IN_MOVE_SELF	The watched file or directory was itself moved.
IN_MOVED_FROM (+)	Generated for the directory containing the old file name when a file is renamed.
IN_MOVED_TO (+)	Generated for the directory containing the new file name when a file is renamed.
IN_OPEN (*)	File or directory was opened.

The following bits may be set in the **mask** field that is returned by an inotify read call.

Event	Description
IN_IGNORED	Watch was removed explicitly or automatically when a file was deleted or the file system was unmounted.
IS_ISDIR	Subject of this event is a directory.
IN_Q_OVERFLOW	Event queue overflowed ( <b>wd</b> is -1 for this event).
IN_UNMOUNT	File system containing the watched object was unmounted.

When a directory is monitored, the following events occur.

- The events marked with an asterisk (\*) can occur both for the directory itself and for objects inside the directory.
- The events marked with a plus sign (+) occur only for objects inside the directory, not for the directory itself.

## Usage notes

1. The **inotify\_init** service creates an inotify instance and returns a file descriptor that refers to the inotify instance. It also allows a **flags** parameter for blocking and nonblocking read and close-on-exec to allow sub processes from reading the inotify file descriptor.

## Related services

- [“inotify\\_add\\_watch \(BPX1IAW, BPX4IAW\) — Add or modify a watch to an initialized inotify instance” on page 338](#)
- [“inotify\\_rm\\_watch \(BPX1IRW, BPX4IRW\) — Remove existing watches from inotify instances” on page 344](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1INI \(inotify\\_init\) example” on page 1253](#) and [“BPX4INI \(inotify\\_init\) example” on page 1346](#).

# inotify\_rm\_watch (BPX1IRW, BPX4IRW) — Remove existing watches from inotify instances

## Function

The **inotify\_rm\_watch** service removes the watch associated with the watch descriptor that is returned by the call to **inotify\_add\_watch** from the inotify instance associated with the file descriptor. Removing a watch causes an IN\_IGNORED event to be generated for this watch descriptor.

For the macro, see [“BPXYINFY— Map inotify\\_init syscall parameters” on page 1080](#).

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task

Operation	Environment
Cross memory mode	PASN = HASN
AMODE (BPX1EFD)	31-bit
AMODE (BPX4EFD)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1IRW(File_Descriptor,
             Watch_Descriptor,
             Return_value,
             Return_code,
             Reason_code)
```

AMODE 64 callers use BPX4IAW with the same parameters.

## Parameters

### File\_Descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The inotify file descriptor is an integer that represents an instance of the inotify returned by the **inotify\_init** call.

### Watch\_Descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The file descriptor is an integer that represents a watched item of the path name that is returned by the **inotify\_add\_watch** call.

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

On success, **inotify\_rm\_watch** returns zero. On error, -1 is returned and Return\_code is set to indicate the cause of the error.

### Return\_code

Returned parameter.

**Type**

Integer

**Length**

Fullword

When the return\_value is -1, the following return codes are possible.

Return_code	Explanation
<b>EBADF</b>	File_Descriptor is not valid.
<b>EINVAL</b>	The Watch_Descriptor is not valid, or File_descriptor is not an inotify file descriptor.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

A fullword in which the **inotify\_rm\_watch** service stores the reason code. The Reason\_code is set only if Return\_value is -1. Reason\_code further qualifies the Return\_value.

**Usage notes**

1. The **inotify\_rm\_watch** service removes a watched term from an inotify watch list. When all file descriptors referring to an inotify instance have been closed using close(), the all resources are freed for reuse by the file system, including all associated watches.

**Related services**

- [“inotify\\_add\\_watch \(BPX1IAW, BPX4IAW\) — Add or modify a watch to an initialized inotify instance” on page 338](#)
- [“inotify\\_init \(BPX1INI, BPX4INI\) — Initialize an inotify instance” on page 341](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1IRM \(inotify\\_rm\\_watch\) example” on page 1254](#) and [“BPX4IAW \(inotify\\_add\\_watch\) example” on page 1346](#).

## IPCSDumpAccess (BPXGMPTR, BPXGMPT4) — PTRACE IPCS dump access service

---

**Function**

The IPCSDumpAccess service reads storage, registers, program attributes, and other information related to a process or thread in a dump that has been opened with the IPCSDump Open/Close service (BPXGMCDE, BPXGMCD4).



## Requirements

Operation	Environment
<b>Authorization:</b>	Problem state, user PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPXGMPTR):</b>	31-bit
<b>AMODE (BPXGMPT4):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPXGMPTR, (Dcor_Request,
                OpenToken,
                Parm1Address,
                Parm2Address,
                Parm3Address,
                Return_value,
                Return_code,
                Reason_code)
```

AMODE 64 callers use BPXGMPT4 with the same parameters.

## Parameters

### Dcor\_Request

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the integer value for the function requested. The functions are explained in the usage notes. The request integer values are defined in the BPXYDCOR macro. See [“BPXYDCOR — dbx cordump cache information” on page 1069](#).

### OpenToken

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the open token that was returned by the BPXGMCDE (BPXGMCD4) Dcor\_Open request.

### Parm1Address

Supplied parameter

#### Type:

Address or Integer

**Length:**

Variable

The name of a required value that contains the first parameter described by the function requested. See [“BPXYDCOR — dbx cordump cache information” on page 1069](#).

**Parm2Address**

Supplied parameter

**Type:**

Address or Integer

**Length:**

Variable

The name of a required value that contains the second parameter described by the function requested. If a second parameter is not required, this value may be zero. See [“BPXYDCOR — dbx cordump cache information” on page 1069](#).

**Parm3Address**

Supplied parameter

**Type:**

Address or Integer

**Length:**

Variable

The name of a required value that contains the third parameter described by the function requested. If a third parameter is not required, this value may be zero. See [“BPXYDCOR — dbx cordump cache information” on page 1069](#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the IPCSDumpAccess service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The address of a fullword in which the IPCSDumpAccess service stores the return code. The IPCSDumpAccess service returns Return\_code only when the Return\_value is -1 and the Reason\_code is DcorPTR\_RsnDcorError. The IPCSDumpAccess service can return one of the following values in the Return\_code parameter:

**Dcor\_PTRrc\_OK**

The specified function completed successfully.

**Dcor\_PTRrc\_AsidNotFound**

An address space could not be found in the dump to satisfy this request.

**Dcor\_PTRrc\_AsidNotSet**

An ASID or PID has not been established for this session.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the IPCSDumpAccess service stores the reason code. The IPCSDumpAccess service returns Reason\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. The following reason codes can accompany the return code:

**RsnOkValue**

The specified function completed successfully.

**RsnDcorError**

See Dcor return codes.

**RsnMVSError**

A getmain error probably occurred.

**RsnIPCSError**

Use the log data set to determine the cause of the error.

**Usage notes**

Table 6 on page 349 shows the constant options you can select for the Dcor\_Request parameter. See [“BPXYDCOR — dbx cordump cache information”](#) on page 1069.

Table 6. Dcor_Request options	
Function request	Explanation
<b>Dcor_ASID_LIST#</b>	Return a list of ASIDs, and the number of ASIDs contained in the list. This list is described in BPXYDCOR as the AsidList_Map.
<b>Dcor_SET_ASID#</b>	Set the current address space ID to view in the dump. If a null parameter is provided, the home address space at the time the dump was taken will be returned. Changing the ASID may alter other values, such as the PID or the current thread.
<b>Dcor_PID_LIST#</b>	Return a list of PIDs, and the number of PIDs contained the list. This list is described in BPXYDCOR as the PidList_Map.
<b>Dcor_SET_PID#</b>	Set the current process ID to view in the dump. If a null parameter is provided, the active process at the time the dump was taken will be returned. The process requested must exist in the current address space. Changing the PID may also cause the current thread to change.
<b>Dcor_LDINFO#</b>	Return the loader data from the current thread.
<b>Dcor_THREAD_LIST#</b>	Return the list of threads contained in the current PID, and the number of threads in the list. The Thread_list mapping is described in BPXYPTRC as PtPxInfo.
<b>Dcor_THREAD_CURRENT#</b>	Return the value of the current thread.
<b>Dcor_SET_THREAD#</b>	Set the current thread ID to view in the dump.
<b>Dcor_PSW#</b>	Return the 16-byte PSW for the current thread.
<b>Dcor_ERROR_PSW#</b>	Return the 16-byte PSW that caused the dump to be taken.

<i>Table 6. Dcor_Request options (continued)</i>	
Function request	Explanation
<b>Dcor_GPR_LIST#</b>	Return the 64-bit GPRs for the current thread.
<b>Dcor_ERROR_GPR_LIST#</b>	Return the 64-bit GPRs active at the time of error.
<b>Dcor_FLT_LIST#</b>	Return the 64-bit FLTs for the current thread.
<b>Dcor_ERROR_FLT_LIST#</b>	Return the 64-bit FLTs active at the time of error.
<b>Dcor_THREAD_STATUS#</b>	Return the Thread_list entry of the current thread. The Thread_list mapping is described in BPXYPTRC as PtPxInfo.
<b>Dcor_READ_D#</b>	Retrieve dump data and place it in a buffer provided by the caller.
<b>Dcor_CAPTURE#</b>	Return the address of a buffer containing the requested dump data.
<b>Dcor_CONDINFO#</b>	Return the current abend information at the time of error. CondInfo is described in BPXYDCOR by the CondInfo_Map.
<b>Dcor_IPCSCMD#</b>	Pass a command to IPCS. The output is stored in an MVS sequential data set named 'userid.BPXGCOR.IPCSPRNT'.

Table 7 on page 350 shows the PTRACE service options for the Dcor\_Request parameter. For each option, the meanings of the Parm1, Parm2, and Parm3 parameters are shown. The terms in the table are described in “BPXYDCOR – dbx cordump cache information” on page 1069.

<i>Table 7. PTRACE service options for the Dcor_Request parameter</i>			
Function request	Parm1	Parm2	Parm3
<b>Dcor_ASID_LIST#</b>	The address of a fullword location to receive the list address	The address of a fullword location to receive the count of ASIDs	0
<b>Dcor_SET_ASID#</b>	The address of a 16-bit location that contains 0 or an ASID value	0	0
<b>Dcor_PID_LIST#</b>	The address of a fullword location to receive the list address.	The address of a fullword location to receive the count of PIDS.	0
<b>Dcor_SET_PID#</b>	The address of a fullword location that contains 0 or a PID value.	0	0
<b>Dcor_LDINFO#</b>	The address of a fullword location to receive the address of the loader information	0	0

<i>Table 7. PTRACE service options for the Dcor_Request parameter (continued)</i>			
Function request	Parm1	Parm2	Parm3
<b>Dcor_THREAD_LIST#</b>	The address of a fullword location to receive the address of the GPR list	0	0
<b>Dcor_THREAD_CURRENT#</b>	The address of an 8-byte location that contains a null value or a thread ID	0	0
<b>Dcor_SET_THREAD#</b>	The address of an 8-byte location that contains a null value or a thread ID	0	0
<b>Dcor_PSW#</b>	The address of a 16-byte location to receive the PSW	0	0
<b>Dcor_ERROR_PSW#</b>	The address of a 16-byte location to receive the PSW	0	0
<b>Dcor_GPR_LIST#</b>	The address of a fullword location to receive the address of the GPR list	The address of a fullword location to receive the length of the GPR list	0
<b>Dcor_ERROR_GPR_LIST#</b>	The address of a fullword location to receive the address of the GPR list	The address of a fullword location to receive the length of the GPR list	0
<b>Dcor_FLT_LIST#</b>	The address of a fullword location to receive the address of the FLT list	The address of a fullword location to receive the length of the FLT list	0
<b>Dcor_ERROR_FLT_LIST#</b>	The address of a fullword location to receive the address of the FLT list	The address of a fullword location to receive the length of the FLT list	0
<b>Dcor_THREAD_STATUS#</b>	The address of a fullword location to receive the address of the thread list	The address of a fullword location to receive the length of the thread list	
<b>Dcor_READ_D#</b>	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) input value of virtual storage in the dump	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) input value of the number of bytes to return	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address that contains the address of the caller's input buffer

<i>Table 7. PTRACE service options for the Dcor_Request parameter (continued)</i>			
<b>Function request</b>	<b>Parm1</b>	<b>Parm2</b>	<b>Parm3</b>
<b>Dcor_CAPTURE#</b>	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) value of virtual storage in the dump	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) value of the number of bytes to return	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address to return the output buffer address
<b>Dcor_CONDINFO#</b>	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address to return the address of the CondInfo data	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address to return the length of the CondInfo data	0
<b>Dcor_IPCSCMD#</b>	The address of the text of an IPCS command, or the word 'LOG'. LOG causes the diagnostic information that is active in the session to be written to an MVS sequential output data set.	A fullword input value, of the length of the IPCS command.	A fullword input value that contains the LRECL to use when allocating the sequential MVS data set userid.BPXGCORE.IPCSPRN T, which contains the output of the IPCS command.

### Related services

- [“IPCSDumpOpenClose \(BPXGMCDE, BPXGMCD4\) — MVS IPCS dump open/close service” on page 352](#)

### Characteristics and restrictions

None.

## IPCSDumpOpenClose (BPXGMCDE, BPXGMCD4) — MVS IPCS dump open/close service

### Function

The IPCSDumpOpenClose service opens (and closes) a dump that has been captured with an SVC dump, a SYSMDUMP, or the DUMP command. Once the dump has been opened, it can be processed with the BPXGMPTR callable service, which reads storage, registers, program attributes, and other dump-related information.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Problem state, user PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPXGMCDE)</b>	31-bit
<b>AMODE (BPXGMCD4)</b>	64-bit

<b>Operation</b>	<b>Environment</b>
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPXGMCDE, (Dcor_Open,
                LevelIndicator,
                DumpDataSetName,
                LogDataSetName,
                ExecDataSetName,
                ClistDataSetName,
                DDIRStr,
                ErrorMessagePtr,
                Return_code,
                Return_Value1,
                Return_Value2,
                Return_Value3)

CALL BPXGMCDE, (DCOR_Close,
                OpenToken)
```

AMODE 64 callers use BPXGMCD4 with the same parameters.

## Parameters

### Dcor\_Open

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the constant for an open request, DCOR\_OPEN. The value of this constant is defined in the BPXYDCOR macro (see [“BPXYDCOR — dbx cordump cache information” on page 1069](#)). If the open request is completed successfully, the BPXGMCDE (BPXGMCD4) service returns a nonzero open token in register 15. This token is used by the BPXGMCDE (BPXGMCD4) close function and the BPXGMPTR (BPXGMPT4) callable service.

If the open request is not successful, the BPXGMCDE (BPXGMCD4) service returns a token value of zero, with explicit failure information in the Return\_code, Return\_value1, Return\_value2, and Return\_value3 fields.

### LevelIndicator

Parameter supplied and returned.

#### Type

Address

#### Length

Fullword

The name of a fullword that contains the release level of the DCOR services. The level number is defined in BPXYDCOR.

### DumpDataSetName

Supplied parameter.

## Type

Character string

## Length

Variable

The name of a required null (X'00'-terminated) character string that provides the name of the dump that is to be opened. The name can be an MVS data set name or a z/OS UNIX file name. An MVS data set name must begin with a double slash (//); otherwise, the name is considered to be the name of a file. To indicate that an MVS data set name is fully qualified, quotation marks should be used on each side of the data set name (// 'MVS.DATA.SET '). When quotation marks are not used to fully qualify the data set name, the login user ID is prefixed to the data set name (userid.MVS.DATA.SET).

## LogDataSetName

Supplied parameter

## Type

Character string

## Length

Variable

The name of an optional null (X'00'-terminated) character string that provides the name of a log data set. The name must be an MVS sequential data set name; a z/OS UNIX file cannot be used as a log data set. The data set name is considered to be fully qualified; quotation marks may be used but they are not necessary.

TSO messages that are generated from running IPCS are written to the log data set. This log is useful in problem determination, especially when the IPCS environment does not get established.

## ExecDataSetName

Supplied parameter.

## Type

Character string

## Length

Variable

The name of an optional null (X'00'-terminated) character string that provides the name of the MVS PDS data set that is to be used in place of SYS1.SBPXEXEC, which is the default. SYS1.SBPXEXEC contains the REXX exec BPXTIPCS, which is used to create a dump directory and establish the IPCS environment.

The name must be an MVS data set name; z/OS UNIX files are not supported for this parameter. The data set name is considered to be fully qualified: Quotation marks can be used, but they are not necessary.

## ClistDataSetName

Supplied parameter.

## Type

Character string

## Length

Variable

The name of an optional null (X'00'-terminated) character string that provides the name of the MVS PDS data set that is to be used in place of SYS1.SBLSCLIO, which is the default. SYS1.SBLSCLIO contains IPCS CLISTs, including BLSCDDIR, which is used to allocate a temporary or permanent dump directory.

The name must be an MVS data set name; z/OS UNIX files are not supported for this parameter. The data set name is considered to be fully qualified: Quotation marks can be used, but they are not necessary.



**DDIRStr**

Supplied parameter.

**Type**

Character string

**Length**

Variable

The name of an optional null (X'00'-terminated) character string that is used to tailor the use of the IPCS dump directory on the invocation of the BLSCDDIR command. It may contain any of the parameters that are accepted by BLSCDDIR. (For more information about BLSCDDIR, see [BLSCDDIR CLIST - create a dump directory in z/OS MVS IPCS Commands](#).)

The name must be an MVS data set name; z/OS UNIX files are not supported for this parameter. The data set name is considered to be fully qualified: quotation marks can be used, but they are not necessary.

The BLSCDDIR command uses a VOLSER of VSAM01 to allocate a new dump directory if the VOL parameter is not provided here.

**ErrorMsgPtr**

Returned parameter.

**Type**

Character string

**Length**

Variable (Fullword)

The name of a required fullword area that will be set to point to a string terminated by a null (X'00'-terminated) character string containing one or more messages that describe certain types of errors that can occur. This string can be sent to a standard error device or file by the caller, and used to inform the end user of the specific reasons for certain failures. If no messages are returned, the string is the null character. Not all error cases return a message string.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the IPCSDumpOpenClose service returns the reason for the failure of the open request. This field is meaningful only when the open token returned in register 15 is 0. The value and meaning of Return\_value1, Return\_value2, and Return\_value3 are dependent upon the nonzero value that is returned in the Return\_code field. For more information about these fields, see ["BPXYDCOR — dbx cordump cache information" on page 1069](#).

<b>Return_code</b>	<b>Explanation</b>
<b>Dcor_CDErc_OK</b>	The specified function completed successfully.
<b>Dcor_CDErc_ParmErr</b>	A parameter error was detected. Return_value1 determines the specific reason for the failure: <ul style="list-style-type: none"> <li>• Dcor_R1_ParmErr_FuncCodeErr. The function code is not supported.</li> <li>• Dcor_R1_ParmErr_DumpDsnReq. The dump data set name is required.</li> </ul>

Return_code	Explanation
<b>Dcor_CDErc_ProcErr</b>	<p>A DCORE processing error occurred. Return_value1 determines the specific reason for the failure:</p> <ul style="list-style-type: none"> <li>• <b>Dcor_R1_ProcErr_SystemErrATC</b> - An unexpected system error occurred while the IPCS environment was being established. Return_value2 contains the ABEND reason code.</li> </ul>
<b>Dcor_CDErc_IKJTSEVErr</b>	The system encountered an error while trying to establish a TSO environment with the IKJTSEV service. See the return values for more information.
<b>Dcor_CDErc_IKJEFTSRErr</b>	The system encountered an error while trying to run the REXX exec with the IKJEFTSR service. See the return values for more information.
<b>Dcor_CDErc_AllocateErr</b>	<p>The system encountered an error while trying to allocate one of the specified data sets. Return_value1 identifies the data set that caused the failure; Return_value2 contains the return code from dynamic allocation (DYNALLOC); and Return_value3 contains the reason code.</p> <ul style="list-style-type: none"> <li>• <b>Dcor_R1_AllocateErr_LogDsn</b>. There was an error allocating the log data set.</li> <li>• <b>Dcor_R1_AllocateErr_ExecDsn</b>. There was an error allocating the EXEC data set.</li> </ul>

**Return\_value1, Return\_value2, and Return\_value3**

Returned parameters.

**Type**

Integer

**Length**

Fullword

The names of fullwords in which the IPCSDumpOpenClose service returns details of the error indicated by Return\_code. See the mapping of BPXYDCOR for detailed information about these fields.

**Dcor\_Close**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The names of a fullword that contains the constant for a close request, DCOR\_CLOSE. The value of this constant is defined in the BPXYDCOR macro see [“BPXYDCOR — dbx cordump cache information” on page 1069](#)).

**OpenToken**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the open token returned by the BPXGMCDE (BPXGMCD4) open request.

## Usage notes

1. The routine to be executed receives control with the following attributes:
  - Problem program state.
  - Key of the target pthread task.
  - AMODE 31.
  - Primary ASC mode.
2. The register usage on entry to the specified routine is:
  - R0: Undefined.
  - R1: Address of the parameter list, as specified by the caller of the IPCSDumpOpenClose service.
  - R2-R12: Undefined.
  - R13: Address of a 72-byte work area in the key that the routine gains control under.
  - R14: The return address from the specified routine to the IPCSDumpOpenClose service. This address must be preserved by the invoked routine.
  - R15: Address of the invoked routine.
3. Only tasks that are created with pthread\_create or the IPT can invoke this service. If a task that is not an IPT or a pthread-created task requests this service, it receives an EACCES return code.
4. At any given time only one pthread can have this service request pending for a given target pthread. If a pthread requests this service for a given target pthread when another pthread already has this service pending for that target pthread, the last pthread receives an EAGAIN return code. It is the caller's responsibility to serialize the invocation of IPCSDumpOpenClose, or contain retry logic for cases in which the EAGAIN return code is received.
5. The EXITRTN assembler routine cannot issue callable services after it gains control under the target pthread.
6. The specified routine can establish its own recovery environment. However, even if recovery is not established, the IPCSDumpOpenClose service establishes its own recovery environment while running under the target pthread. For all recoverable errors, this recovery routine retries, returning the EFAULT return code to the requester. It also ensures that any recovery routine that is established by the target pthread itself is not entered unexpectedly.

## Related services

- [“IPCSDumpAccess \(BPXGMPTR, BPXGMPT4\) — PTRACE IPCS dump access service” on page 346](#)

## Characteristics and restrictions

None.

# isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal

---

## Function

The isatty callable service determines whether a file is a terminal.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task

Operation	Environment
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE:</b>	31-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1ITY, (File_descriptor,  
              Return_value)
```

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword containing the file descriptor.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the isatty service returns 1 if File\_descriptor is a terminal, or 0 if it is not a terminal.

## Usage notes

This function does not return -1. If the file descriptor is not valid, a zero is returned. If this service fails for other reasons, the process abends.

## Related services

- [“ttyname \(BPX1TYN, BPX4TYN\) \(POSIX version\) — Get the name of a terminal ” on page 956](#)
- [“isatty \(BPX2ITY, BPX4ITY\) \(X/Open Version\) — Determine whether a file descriptor represents a terminal” on page 359](#)

## Characteristics and restrictions

None.

**Examples**

See “BPX1ITY (isatty) example” on page 1254.

## isatty (BPX2ITY, BPX4ITY) (X/Open Version) – Determine whether a file descriptor represents a terminal

---

**Function**

The isatty callable service determines whether a file is a terminal.

**Requirements**

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX2ITY)</b>	31-bit
<b>AMODE (BPX4ITY)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX2ITY,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ITY with the same parameter.

**Parameters****File\_descriptor**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

## kill (BPX1KIL, BPX4KIL)

The name of a fullword in which the isatty service returns 1 if File\_descriptor is a terminal, or 0 if it is not a terminal.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the isatty service stores the return code. The isatty service may return Return\_code only if Return\_value is 0. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The isatty service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EBADF	The filedes argument is not a valid open file descriptor.
ENOTTY	The filedes argument is not associated with a terminal.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the isatty service stores the reason code. The isatty service may return Reason\_code only if Return\_value is 0. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. This version of isatty is XPG4 compliant.
2. This function does not return -1. If the file descriptor is not valid, a zero is returned.

## Related services

- [“ttyname \(BPX2TYN, BPX4TYN\) \(X/Open version\) — Get the name of a terminal”](#) on page 958
- [“isatty \(BPX1ITY\) \(POSIX Version\) — Determine whether a file descriptor represents a terminal”](#) on page 357

## Characteristics and restrictions

None.

## Examples

See [“BPX2ITY \(isatty\) example ”](#) on page 1254.

## kill (BPX1KIL, BPX4KIL) — Send a signal to a process

---

### Function

The kill callable service sends a signal to a process, a process group, or all processes in the system to which the caller has permission to send a signal.

**CAUTION:**

When a caller with appropriate privileges (see [“Authorization”](#) on page 8) specifies a pid equal to -1, the signal will normally be sent to all processes in the system, excluding the init process (process ID 1). If the signal action is to terminate the process, all processes will terminate. This may not be the desired action, considering that some processes may be necessary for the continued operation of the system.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1KIL):</b>	31-bit
<b>AMODE (BPX4KIL):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax diagram is as follows:

```
CALL BPX1KIL, (Process_ID,
               Signal,
               Signal_Options,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4KIL with the same parameters.

**Parameters****Process\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword whose value specifies the process or processes to which the signal is to be sent:

- If Process\_ID is greater than 0, it is assumed to be a process ID. The signal is sent to the process with that specific process ID.
- If Process\_ID is equal to 0, the signal is sent to all processes with a process group ID equal to that of the caller, and for which the caller has permission to send a signal.
- If Process\_ID is -1, the signal is sent to all processes for which the caller has permission to send the signal.

- If Process\_ID is less than -1, its absolute value is assumed to be a process group ID. The signal is sent to all processes with a process group ID equal to that absolute value, and for which the caller has permission to send a signal.

For more information, see [“Characteristics and restrictions”](#) on page 364.

**Signal**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the signal number to be sent to the processes that are indicated by the Process\_ID parameter. The signal number must be defined in the BPXYSIGH macro, or 0.

If the signal is 0, error checking takes place, but no signal is sent. Use a signal value of 0 to verify that the Process\_ID parameter is correct before actually sending a signal. This method does not verify permission to send the signal to the specified Process\_ID.

**Signal\_Options**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword that contains the binary flags that describe how the signal is to be handled by z/OS UNIX and the user-supplied signal interface routine (SIR). This byte of user information is passed to the SIR in a data structure that is mapped by the BPXYPPSD macro. See [“BPXYPPSD — Map signal delivery data”](#) on page 1124. Signal\_Options are mapped as follows:

**First 2 bytes**

User-defined bytes that are delivered with the signal to the SIR in the signal information control block. These bytes are mapped by PPSDKILDATA.

**Last 2 bytes**

Flag bits, mapped by PPSDKILOPTS, that are defined as follows:

- First bit - signal to bypass Ptrace processing
- Second bit - reserved
- Third bit - signal code specified in first 2 bytes, set by the application
- Fourth bit - reserved
- Fifth bit - enhanced SIGKILL deliverability (superkill)
- Sixth bit - Indicates whether to override the SIGTRACE action. If this bit is ON, then the seventh bit indicates the override action. If this bit is OFF, the SIGTRACE action is to toggle the current user syscall trace setting for the target processes.
- Seventh bit - If this bit is ON, the SIGTRACE signal turns user syscall tracing ON. If OFF, then SIGTRACE turns user syscall tracing OFF.
- Remaining bits - reserved.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword



The name of a fullword in which the kill service returns 0 if it has permission to send the specified signal to any of the processes specified by the Process\_ID parameter. A return value of 0 means that a signal was sent (or could have been sent, if the signal value was 0) to at least one of the specified processes.

If no signal is sent, -1 is returned.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the kill service stores the return code. The kill service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The kill service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value of Signal is incorrect, or is not the number of a supported signal.
<b>EMVSAF2ERR</b>	The caller does not have the appropriate RACF permissions.
<b>EPERM</b>	The caller does not have permission to send the signal to any process that was specified by the Process_ID parameter.
<b>ESRCH</b>	No processes or process groups that correspond to Process_ID were found.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the kill service stores the reason code. The kill service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSAF2ERR, the reason code contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the Check Privilege service return and reason code values for RACF, see the following table:

RACF return code	RACF reason code	Explanation
8	4	The caller is not the owner of the target process.
8	12	There was an internal error during security product processing.

### Related services

- [“getpid \(BPX1GPI, BPX4GPI\) — Get the process ID” on page 298](#)
- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ” on page 777](#)
- [“setsid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ” on page 793](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)

## Characteristics and restrictions

1. A caller can send a signal if the real or effective user ID of the caller is the same as the real or saved set user ID of the intended recipient. A caller can also send signals if it has appropriate privileges.  
Permissions are discussed in [“Authorization” on page 8](#).
2. Regardless of user ID, a caller can always send a SIGCONT signal to a process that is a member of the same session as the sender.
3. A caller can also send a signal to itself. If the signal is not blocked, at least one pending unblocked signal is delivered to the sender before the service returns control. Provided that no other unblocked signals are pending, the signal delivered is the signal sent. See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413](#) for more information.
4. The thread-scoped signals (SIGTHSTOP and SIGTHCONT) cannot be issued by the kill callable service. They can be issued only by the pthread\_kill service.
5. An enhanced SIGKILL (superkill) can be sent by setting the PPSDSUPERKILL bit on in the BPXYPPSD (“BPXYPPSD — Map signal delivery data” on page 1124). The superkill will break through most of the signal deterrents that can be an obstacle to the normal delivery of a SIGKILL and the resulting termination of the target process.

### Note:

- a. You cannot use pthread\_kill() or sigqueue() to do a superkill. The superkill option is ignored for these services.
- b. You cannot do a superkill to a group, or specify a PID of -1 (kill everyone).
- c. When a target process has blocked all signals with the set\_dub\_default (BPX1SDD/BPX4SDD) service, superkills are deferred. The kill does not fail; it is simply ignored by the target process.
- d. Before a process can be ended by a superkill, a regular SIGKILL must be sent to it, or the attempt will result in EINVAL/JRSigkillNotSent. This is analogous to the required cancel before a force arm.

If the environment is valid, the target process ends with a X'422' abend and reason code X'0109'.

## Examples

See [“BPX1KIL \(kill\) example” on page 1254](#).

## MVS-related information

For signal information, see [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413](#).

# **\_\_login, \_\_login\_\_applid, \_\_certificate (BPX1SEC, BPX4SEC) — Provides an interface to the security product**

---

## Function

The BPX1SEC/BPX4SEC callable service provides an interface to the security product to allow the calling process to obtain security-related services.

No special authority is required to use this service to register or unregister a certificate that has the current identity of the calling process.

The C functions \_login and \_certificate result in a call to this service.

## Requirements

### Operation

### Environment

### Authorization:

Supervisor state or problem state, any PSW key

Operation	Environment
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SEC):</b>	31-bit
<b>AMODE (BPX4SEC):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SEC, (Function_code,
               Identity_type,
               Identity_length,
               Identity,
               Pass_length,
               Pass,
               Certificate_length,
               Certificate,
               Option_flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SEC with the same parameters.

## Parameters

### Function\_code

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that specifies a numeric value identifying the function that is to be performed. The following Function\_code constants are defined by the BPXYCONS macro. See [“BPXYCONS — Constants used by services” on page 1062](#).

Constant	Description
<b>SECURITY_CREATE#</b>	Create the security environment for the caller's process.
<b>SECURITY_CERTREG#</b>	Register the passed certificate with the user ID that is associated with the current security environment.
<b>SECURITY_CERTDEREG#</b>	Deregister the passed certificate from the user ID that is associated with the current security environment.
<b>SECURITY_CERTAUTH#</b>	Authenticate the passed certificate for the caller. The certificate must have been registered.

### Identity\_type

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that identifies the format of the Identity parameter and the Pass parameter. Constants are defined by the BPXYCONS macro. See [“BPXYCONS — Constants used by services” on page 1062](#).

**Constant****Description****SECURITY\_USERID#**

The user identity is in the format of a 1-to 8-character user ID that is passed as input.

**Identity\_length**

Supplied parameter for SECURITY\_USERID#

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Identity parameter. The specified length must be consistent with the allowable Identity types: for SECURITY\_USERID#, the length is 1-to-8 characters.

**Identity**

Supplied parameter for SECURITY\_USERID#

**Type:**

Character string

**Character set:**

For SECURITY\_USERID#, the identity is a USERID that follows the XPG4 naming convention portable character set. This includes upper and lower-case letters (A-Z, a-z), numerics (0–9), period (.), dash (-), and underbar (\_).

**Length:**

Specified by the Identity\_length parameter

The name of a field that contains the user identity in the specified format.

**Pass\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket or between 9 and 100 characters for a password phrase. A length of zero indicates that the Pass parameter is to be ignored.

**Pass**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pass\_length parameter

The name of a field, of length Pass\_length, that contains, left-justified, the password, PassTicket or password phrase that is to be verified.

**Certificate\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

For SECURITY\_CERTREG#, SECURITY\_CERTDEREG#, and SECURITY\_CERTAUTH#, the name of a fullword that contains the length of a certificate structure as defined by the Certificate parameter. This parameter is ignored for all other function codes.

**Certificate**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Variable

For SECURITY\_CERTREG#, SECURITY\_CERTDEREG#, and SECURITY\_CERTAUTH#, the name of an area that consists of a digital certificate. For a description of the formats for a digital certificate, see [initACEE \(IRRSIA00\) callable service in z/OS Security Server RACF Callable Services](#). This parameter is ignored for all other function codes.

**Option\_flags**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword binary field that contains the BPX1SEC/BPX4SEC options. If no options are required, specify the name of a fullword field that contains 0. No options are currently defined.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the BPX1SEC/BPX4SEC service returns 0 if the request is successful, or -1 if it is not successful. For SECURITY\_CERTAUTH#, this field returns an address to read-only storage that contains the 8-character user ID. If the request is not successful, the service returns -1.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the BPX1SEC/BPX4SEC service stores the return code. The BPX1SEC/BPX4SEC service returns Return\_code only if Return\_value is -1. For a list of return code values, see

Return codes (errno) in *z/OS UNIX System Services Messages and Codes* The BPX1SEC/BPX4SEC service can return one of the following values in the Return\_code parameter:

Return_Code	Explanation
<b>EINVAL</b>	A parameter is not valid, or a certificate was not specified, or no security product is installed. The following reason codes can accompany the return code: JrFunctionCode, JrIdentityType, JrBadOptions, JrUserNameLenError, JrPasswordLenError, JrNewPasswordLenError, JrCertificate, JrNoSecurityProduct.
<b>EPERM</b>	The operation is not permitted, or the calling task has a task level ACEE that was not created by a prior call to this service. The following reason codes can accompany the return code: JrNotServerAuthorized, JrSecurityEnv, JrEnvDirty, JrMultiThreaded, JrUnexpectedError.
<b>ESRCH</b>	The USERID cannot become an OMVS process. The following reason codes can accompany the return code: JrOK, JrNoCertforUser.
<b>EMVSSAF2ERR</b>	An error occurred in the security product, or there was a parameter list error on a call to initACEE. The following reason codes can accompany the return code: JrCertInvalid, JrSafInternal, JrSafGroupNoMVS, JrSafNoGid, JrSafNoUid, JrSafUserNoMVS, JrSafParmListErr, JrCertInvalid, JrCertDoesNotMeetReq, JrCertAlreadyDefined, JrUnexpectedError, JrOK, X'0814'.
<b>ENOSYS</b>	The function is not implemented. The following reason codes can accompany the return code: JrNoSecurityProduct, JrNoInitACEE.
<b>EACCES</b>	Permission is denied. The following reason codes can accompany the return code: JrOK, JrNoResourceAccess.
<b>EMVSEXPIRE</b>	The password for the resource that was specified has expired. The following reason code can accompany the return code: JrOK.
<b>EMVSPASSWORD</b>	The new password that was specified is not valid. The following reason code can accompany the return code: JrOK.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which this service stores the reason code. The BPX1SEC/BPX4SEC service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

Table 8 on page 368 shows the Return\_code and Reason\_code values that are returned when the BPX1SEC/BPX4SEC service is called to register or unregister a certificate and initACEE has a return code of 8.

Table 8. BPX1SEC/BPX4SEC return values for certificate registration or deregistration with initACEE return code 8			
initACEE reason code	BPX1SEC/BPX4SEC return code	BPX1SEC/BPX4SEC reason code	Explanation
4	EMVSSAF2ERR	JrSafParmListErr	There was a parameter list error.
8	EMVSSAF2ERR	JrSafInternal	There was an internal RACF error.

Table 8. BPX1SEC/BPX4SEC return values for certificate registration or deregistration with initACEE return code 8 (continued)

initACEE reason code	BPX1SEC/BPX4SEC return code	BPX1SEC/BPX4SEC reason code	Explanation
12	EMVSSAF2ERR	JrSafInternal	RACF recovery environment could not be established.
16	EACCES	JrNoResourceAccess	The user is not authorized.
20	EMVSSAF2ERR	JrCertDoesNotMeetReq	The certificate does not meet RACF requirements.
24	EMVSSAF2ERR	JrCertAlreadyDefined	The certificate is already defined for another user.
36	EMVSSAF2ERR	JrCertInvalid	The certificate is not valid.

## Usage notes

1. Table 9 on page 369 shows the BPX1SEC/BPX4SEC parameters that are used with each function.

Table 9. BPX1SEC/BPX4SEC parameter usage based on function requested

Parameter	Login as a new user	Register a certificate	Unregister a certificate	Authenticate a certificate
Function_Code	_CREATE#	_CERTREG#	_CERTDEREG#	_CERTAUTH#
Identity_Type	SECURITY_USERID#	Not applicable	Not applicable	Not applicable
Identity_Length	Input	Not applicable	Not applicable	Not applicable
Pass_Length	Input (optional)	Not applicable	Not applicable	Not applicable
Pass	Input (optional)	Not applicable	Not applicable	Not applicable
Cert_Length	Not applicable	Input	Input	Input
Certificate	Not applicable	Input	Input	Input
Option_Byte	Not applicable	Not applicable	Not applicable	Not applicable
Return_value	Output	Output	Output	Output (address)
Return_code	Output	Output	Output	Output
Reason_code	Output	Output	Output	Output

For the SECURITY\_CERTREG# and SECURITY\_CERTDEREG# functions, the certificate is passed in the Certificate parameter, and not the Identity parameter. The certificate does not necessarily define the identity of the caller; these functions could be called with a user ID and password.

For the SECURITY\_CERTAUTH# function, the certificate is passed in the Certificate parameter. The certificate contains the identity of the caller, and can be used instead of a user ID/password combination.

2. When this service is called for function code SECURITY\_CERTAUTH#:

- A certificate is passed for authentication. It is possible that the USERID associated with the certificate was valid at the time the certificate was created, but is no longer valid when this call is made. For example, the USERID could have been revoked or its profile could have been changed so that it no longer has any z/OS UNIX System Services information associated with it. However, the original ACEE is still valid and can be cached. Because of this there are different return/reason

codes that are issued for revoked USERIDs and USERIDs with no OMVS segment, depending on the ACEE that is used during this call. That is:

If the USERID has no OMVS segment, when using a cached ACEE you will receive ESRCH and JrOK. When there is no cached ACEE, you will receive EMVSSAF2ERR/JrSafUserNoMVS

If the USERID has been revoked, when using a cached ACEE you will receive ESRCH/JrOK. When there is no cached ACEE you will receive EMVSSAF2ERR/JrOK.

If the USERID has an OMVS segment, but no UID defined, when using a cached ACEE you will receive EMVSSAF2ERR/X'0814'. When there is no cached ACEE you will receive EMVSSAF2ERR/JrSafNoUid.

3. Mixed case passwords and PassTickets are supported when the installed security product (such as RACF) supports mixed case; otherwise, passwords and PassTickets are folded to uppercase. Non-graphic characters are always folded to blanks.

The contents of the password phrase string are passed unchanged to the installed security product.

Although z/OS UNIX System Services supports password phrases that are 9-100 characters in length, your installation or the installed security product can have additional rules for password phrase lengths. Ask your security administrator or system programmer if any additional rules apply.

4. The BPX1SEC/BPX4SEC service allows a process to assume an identity that is different from that of the address space. It is assumed that the process will either terminate or select a new user ID, but not try to revert back to the original address space identity. The user could issue the BPX1SEC/BPX4SEC request again with the original user identity; however, at this point the user has its own security environment, at the task level, rather than the address space level.
5. For SECURITY\_CREATE# and SECURITY\_CERTAUTH#, if BPX.DAEMON is defined, then the address space must be program-controlled.
6. When calling the BPX1SEC/BPX4SEC service with function code SECURITY\_CREATE#, the caller can change identities under any of the following conditions:
  - The caller specifies the password for the requested identity.
  - If no password is specified and the BPX.DAEMON profile is not defined in the FACILITY class, the caller must be a superuser. If no password is specified and the caller has read access to the BPX.SRV.userid SURROGAT profile, where *userid* is the user ID specified in the User\_name parameter.
  - If no password is specified and the BPX.DAEMON profile is defined in the FACILITY class, the caller must be permitted to that profile with at least READ access and must be a superuser.
7. Only a single-threaded process can call the BPX1SEC/BPX4SEC service with function code SECURITY\_CREATE#.
8. The purpose of the BPX1SEC/BPX4SEC register/unregister service is to provide a way for the caller to associate or disassociate its user ID with a certificate. No new security environment is created, and no authentication of the user is done.
9. The ability to call the BPX1SEC/BPX4SEC service to register or unregister a certificate with a user ID is not a privileged operation. The user does not need any special authority above that required by RACF to register or unregister certificates. The caller does not, for example, have to be a DAEMON or a SUPERUSER. RACF requires that the caller have access to the RACDCERT FACILITY class definitions (IRR.DIGTCERT.ADD and IRR.DIGTCERT.DELETE) for registration and deregistration.
10. The BPX1SEC/BPX4SEC authenticate service provides the caller with a way to authenticate a security environment using a certificate. The certificate must already be registered. If the certificate is not registered, an error is returned.
11. The \_\_login\_\_applid() function is equivalent to \_\_login() with the added feature that \_\_login\_\_applid() allows an application identifier (applid) to be supplied. The applid is used to verify the user's authority to access the application. When a PassTicket is specified, the applid is also used in conjunction with the USERID to verify the PassTicket. If an application is not using the \_\_login\_\_applid() function but still wants to pass an applid to this service, the application can set the applid value in the BPXYTHLI. Also:



- THLIEP\_FunctionCode is set with ThliEP\_ApplSet.
- THLIEP\_ApplidLen is set to the length of the APPLID. If this value is less than 1 or greater than 8, the ThliEP\_APPLID value is ignored.
- ThliEP\_APPLID is set to the APPLID value.

If there is no applid value passed, the applid value defaults to OMVSAPPL.

12. If environment variable BPXK\_MIN\_PWFOLD=YES is set then non-graphic characters will not be changed to blanks before being passed to the security product. This behavior will exist for all threads in the process where the environment variable was set

## Related services

None.

## Characteristics and restrictions

None.

## Examples

See “BPX1SEC (\_\_login, \_\_login\_\_applid, \_\_certificate) example” on page 1285.

# lchattr (BPX1LCR, BPX4LCR) – Change the attributes of a file or directory or symbolic link

---

## Function

The lchattr service modifies the attributes that are that are associated with a file. It is similar to the chattr service and can be used to modify certain attributes with a symbolic or external link as well as for regular files and directories.

If the path name is a symbolic link, the requested change occurs on the attributes of the symbolic link. Only the attributes that can apply to a symbolic link are updated. These are limited to the owner, the time values, and the security label. All other requested attribute changes have no effect for the symbolic link. See “chattr (BPX1CHR, BPX4CHR) – Change the attributes of a file or directory ” on page 82.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1LCR)</b>	31-bit
<b>AMODE (BPX4LCR)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

AMODE 64 callers use BPX4LCR with the same parameters.

## Format

```
CALL BPX1LCR, (Pathname_length,
               Pathname,
               Attributes_length,
               Attributes,
               Return_value,
               Return_code,
               Reason_code)
```

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the length of the path name of the file whose attributes that you want to change.

### Pathname

Supplied parameter.

#### Type

Character string.

#### Length.

Specified by the Pathname\_length parameter.

The name of a field that contains the path name of the file. The length of this field is specified in Pathname\_length.

If *Pathname* specifies a symbolic link file, the lchattr service changes the attributes of the symbolic link file itself, if the attributes requested can apply to a symbolic link. Only the owner, times, and security label can be changed for a symbolic link. All other attributes do not apply and are ignored.

### Attributes\_length

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the length of the area containing the attributes that you want to change.

### Attributes

Supplied parameter.

#### Type

Structure.

#### Length

Specified by the Attributes\_length parameter.

The name of the area that contains the attributes that you want to change. The area is mapped by BPXYATT. For more information about the content of this area, see [“BPXYATT — Map file attributes for chattr and fchattr”](#) on page 1057.

If *Pathname* specifies a symbolic link file, the lchattr service changes the attributes of the symbolic link file itself, if the attributes requested can apply to a symbolic link. Only the owner, times, and security label can be changed for a symbolic link. All other attributes do not apply and are ignored.

For extended attributes, the area is mapped by the attx structure that is located in the same BPXYATT mapping. Storage for the attribute value must be contiguous with the attx structure and the attributes\_length argument is the length of the attx structure plus the length of the attribute value.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the lchattr service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the lchattr service stores the return code. The lchattr service returns the return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The lchattr service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	<p>The calling process did not have appropriate permissions. Possible reasons include:</p> <ul style="list-style-type: none"> <li>• The calling process was attempting to set access time or modification time to current time and the effective UID of the calling process does not match the owner of the file.</li> <li>• The process does not have write permission for the file or it does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).</li> <li>• The calling process was attempting to truncate the file and it does not have write permission for the file.</li> </ul>
<b>EEXIST</b>	<p>A request to create an extended attribute was specified, but the attribute is already set. The following reason code can accompany the return code: JrAttxExists.</p>
<b>EFBIG</b>	<p>The calling process was attempting to change the size of a file, but the specified length is greater than the maximum file size limit for the process. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRWriteBeyondLimit.</p>
<b>EINVAL</b>	<p>The extended attribute operation is invalid due to the values that are being passed. The input length for the ATTX parameter might be insufficient or the subcommand AttxSubCmd is invalid. The following reason codes can accompany the return code: JRInvMinArgLenAttx, JRInvArgLenAttx, JRInvAttxCmd.</p>
<b>ELOOP</b>	<p>A loop exists in symbolic links that were encountered during resolution of the <i>Pathname</i> argument. This error is issued if more than 24 symbolic links are detected in the resolution of <i>Pathname</i>.</p>

Return_code	Explanation
<b>EMVSERR</b>	An MVS environmental error was detected. The following reason code can accompany the return code: JrSeclabelClassInactive.
<b>ENAMETOOLONG</b>	<i>Pathname</i> is longer than 1023 characters, or a component of the path name is longer than 255 characters. Filename truncation is not supported.
<b>ENOATTR</b>	For the replace or remove function, the specified extended attribute cannot be set. ENOATTR is also set if the passed in attribute name, <i>AttxName</i> , is empty. The following reason codes can accompany the return code: JrAttxName and JrAttxUnavail.
<b>ENOTSUP</b>	The request against an extended attribute is not supported. The following reason codes can accompany the return code: JRInvAttxFTyp, JrSetAttxTrustClas, JRAttxNoSup, and JRSetAttxNameSpc.
<b>ENOENT</b>	No file named <i>Pathname</i> was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOSYS</b>	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.
<b>ENOTDIR</b>	Some component of <i>Pathname</i> is not a directory.
<b>ENOTSUP</b>	The request against an extended attribute is not supported. The following reason codes can accompany the return code: JRInvAttxFTyp, JrSetAttxTrustClas, JRAttxNoSup, and JRSetAttxNameSpc.
<b>EPERM</b>	The operation is not permitted for one of the following reasons: <ul style="list-style-type: none"> <li>• The calling process was attempting to change the mode or the file format, but the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).</li> <li>• The calling process was attempting to change the owner, but it does not have appropriate privileges.</li> <li>• The calling process was attempting to change the general attribute bits, but it does not have write permission for the file.</li> <li>• The calling process was attempting to set a time value (not current time). However, the effective user ID does not match the owner of the file and it does not have appropriate privileges.</li> <li>• The calling process was attempting to set the change time or reference time to current time, but it does not have write permission for the file.</li> <li>• The calling process was attempting to change auditing flags. However, the effective UID of the calling process does not match the owner of the file and the calling process does not have appropriate privileges.</li> <li>• The calling process was attempting to change the security auditor's auditing flags, but the user does not have auditor authority.</li> <li>• The calling process was attempting to set the security label, but one or more of the following conditions apply: <ul style="list-style-type: none"> <li>– The calling process does not have RACF SPECIAL authorization and appropriate privileges.</li> <li>– There is already a security label that is associated with the file.</li> </ul> </li> </ul>
<b>ERANGE</b>	The length of the extended attribute value to be set, <i>AttxValLn</i> , does not match the attribute size itself. For example, ERANGE can occur when a 4-byte attribute is being set to a 2-byte or 6-byte value. The following reason code can accompany the return code: JrInvAttxValLen.

**Return\_code****EROFS****Explanation**

*Pathname* specifies a file that is on a read-only file system. Consult the reason code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the lchattr service stores the reason code. The lchattr service returns a reason code only if the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

<i>Table 10. Attribute fields that can be modified by lchattr</i>		
<b>Set flags</b>	<b>Attribute fields input</b>	<b>Description</b>
<b>ATTMODECHG</b>	ATTMODE	Set the mode according to the value in ATTMODE. See <a href="#">“chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory”</a> on page 101.
<b>ATTOWNERCHG</b>	ATTUID ATTGID	Set the owner user identifier (UID) and group identifier (GID) to the values specified in ATTUID and ATTGID. See <a href="#">“chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory”</a> on page 104.
<b>ATTSETGEN</b>	ATTGENVALUE ATTGENMASK	Only the bits corresponding to the bits set ON in the ATTGENMASK are set to the value (ON or OFF) in ATTGENVALUE. Other bits are unchanged.
<b>ATTTRUNC</b>	ATTSIZE	Change the file size to ATTSIZE bytes. See <a href="#">“ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file”</a> on page 249.
<b>ATTATIMECHG</b>	ATTATIME	If ATTLP64TIMES is not set, set the access time of the file to the value specified in ATTATIME. If ATTLP64TIMES is set, set the access time of the file to the value specified in ATTATIME64, which is a doubleword field.
<b>ATTATIMETOD</b>	None	Set the access time of the file to the current time.

Table 10. Attribute fields that can be modified by lchattr (continued)

Set flags	Attribute fields input	Description
<b>ATTMTIMECHG</b>	ATTMTIME	If ATTLP64TIMES is not set, set the modification time of the file to the value specified in ATTMTIME. If ATTLP64TIMES is set, set the modification time of the file to the value specified in ATTMTIME64, which is a doubleword field.
<b>ATTMTIMETOD</b>	None	Set the modification time of the file to the current time.
<b>ATTMAAUDIT</b>	ATTAUDITORAUDIT	Set the security auditor's auditing flags to the value specified in ATTAUDITORAUDIT. See <a href="#">“chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path”</a> on page 95.
<b>ATTMUAUDIT</b>	ATTUSERAUDIT	Set the user's auditing flags to the value specified in ATTUSERAUDIT. See <a href="#">“chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path”</a> on page 95.
<b>ATTCTIMECHG</b>	ATTCTIME	If ATTLP64TIMES is not set, set the change time of the file to the value specified in ATTCTIME. If ATTLP64TIMES is set, set the change time of the file to the value specified in ATTCTIME64, which is a doubleword field.
<b>ATTCTIMETOD</b>	None	Set the change time of the file to the current time.
<b>ATTREFTIMECHG</b>	ATTREFTIME	If ATTLP64TIMES is not set, set the reference time of the file to the value specified in ATTREFTIME. If ATTLP64TIMES is set, set the reference time of the file to the value specified in ATTREFTIME64, which is a doubleword field.
<b>ATTREFTIMETOD</b>	None	Set the reference time of the file to the current time.
<b>ATTFILEFMTCHG</b>	ATTFILEFMT	Set the file format of the file to the value specified in ATTFILEFMT.
<b>ATTCHARSETIDCHG</b>	ATTFILETAG	Set the file tag. See BPXYSTAT ( <a href="#">“BPXYSTAT — Map the response structure for stat”</a> on page 1162) for file tag mapping.
<b>ATTSECLABELCHG</b>	ATTSECLABEL	Set the initial security label for a file or directory.

1. Flags in the Attributes parameter are set to indicate which attributes are to be updated. To set an attribute, turn the corresponding Set Flag on, and set the corresponding Attributes Field according to [Table 3 on page 87](#). Multiple attributes can be changed at the same time.

Clear the **Set Flag** field before any bits are turned on. It is considered an error if any of the reserved bits in the flag field are turned on.

2. Some of the attributes that are changed by the lchattr service can also be changed by other services. See the related service (listed in [Table 10 on page 375](#)) for a detailed description.
3. Changing mode (ATTMODECHG = ON):

- The file mode field in the Attributes parameter is mapped by the BPXYMODE macro (see [“BPXYMODE — Map the mode constants ” on page 1106](#)). For more information about the values for file type, see [“BPXYFTYP — File type definitions” on page 1078](#).
- File descriptors that are open when the lchattr service is called retain the access permission they had when the file was opened.
- The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
- Setting the set-group-ID-on-execution permission (in mode) means that when this file is run through the exec, attach\_exec, or spawn service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

The set-group-ID-on-execution permission is set to zero if the caller does not have appropriate privileges and the GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.

- Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID. The process will seem to be running under the UID of the file's owner, rather than that of the actual invoker.

4. Changing owner (ATTOWNERCHG = ON):

To change the owner UID of a file, the caller must have appropriate privileges.

To change the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:

- The effective UID of the caller matches the file's owner UID.
- The Owner\_UID value that is specified in the change request matches the file's owner UID.
- The Group\_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
- When the owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
- When the owner is changed, both UID and GID must be specified as they are to be set, or set to -1 if the value is to remain unchanged. If only one of these values is to be changed, the other can be set to its present value or to -1 to remain unchanged.

5. Changing general attribute bits (ATTSETGEN = ON):

- Changing the general attributes of a file, directory, symbolic link, or external link is not supported by BPX1LCR. For more information about setting the general attributes of a file or directory, see [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory ” on page 82](#).

6. Changing the file size (ATTTRUNC = ON):

- The resizing of a file to ATTSIZE bytes changes the file size to ATTSIZE, beginning from the first byte of the file. If the file was originally larger than ATTSIZE bytes, the data from ATTSIZE to the original end of file is removed. If the file was originally shorter than ATTSIZE, bytes between the old and new lengths are read as zeros.

Full blocks are returned to the file system so that they can be used again.

The file offset is not changed.

- When a file size is changed successfully, it clears the set-user-ID, the set-group-ID, and the save-text (sticky bit) attributes of the file, unless the caller has appropriate privileges.
- The resizing of a file to ATTSIZE bytes, where ATTSIZE is greater than the soft file size limit for the process, fails with EFBIG, and the SIGXFSZ signal is generated for the process.

#### 7. Changing times:

- All time fields in Attributes are in POSIX format.
- For the access time or the modification time to be set explicitly (ATTATIMECHG = ON or ATTMTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the access time or modification time to be set to the current time (ATTATIMETOD = ON or ATTMTIMETOD = ON), the effective ID must match the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.
- For the change time or the reference time to be set explicitly (ATTCTIMECHG = ON or ATTREFTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the change time or reference time to be set to the current time (ATTCTIMETOD = ON or ATTREFTIMETOD = ON), the calling process must have write permission for the file.
- For any time field (**atime**, **mtime**, **ctime**, **reftime**), if both current time and specific time are requested (for example, ATTCTIMETOD = ON and ATTCTIMECHG = ON), the current time is set.
- When any attribute field is changed successfully, the file's change time is also updated.

#### 8. Changing auditor audit flags (ATTMAAUDIT = ON):

- For auditor audit flags to be changed, the user must have auditor authority. Users with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

Establish auditor authority by issuing the TSO/E command ALTUSER Auditor.

#### 9. Changing user audit flags (ATTMUAUDIT = ON):

- For the user audit flags to be changed, the user must have appropriate privileges (see [“Authorization” on page 8](#)) or be the owner of the file.

#### 10. Changing file format (ATTFILEFMTCHG = ON):

- The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.

#### 11. Changing the file tag (ATTCHARSETIDCHG=ON):

- A file tag can be set for regular, FIFO, and character special files. If the DeferTag bit is on in the file tag, the file must be empty.

#### 12. Changing the security label (ATTSECLABELCHG=ON):

- For the security label to be changed, the user must have RACF SPECIAL authorization and appropriate privileges (see [“Authorization” on page 8](#)), and no security label must currently exist on the file. Only an initial security label can be set. An existing security label cannot be changed. The function will successfully set the security label if the RACF SECLABEL class is active. If the SECLABEL class is not active, a return code of EMVSERR will be returned.

#### 13. Changing extended attributes:

- To add, remove, or modify extended attributes, use the Extended Attributes control block ATTX in place of the standard ATT control block. For mapping to ATTX, see [“BPXYATT — Map file attributes for chattr and fchattr” on page 1057](#). Rather than turning on flags within the control block like ATT, ATTX supports an attribute name field, AttxName, to specify which attribute to take action against. The ATTX mapping is variable length with the attribute value, AttxVal, field being a CHAR(\*) value. The **AttxValLn** field should be set to the length of the attribute value.



- The subcommand field, **AttxSubcmd**, controls whether this service call will SET or REMOVE the attribute. If GET, LIST, or any other value is specified then the call will fail with JRInvAttrCmd.
- AttxFlags AttxfReplace and AttxfCreate are used to validate the request against the extended attribute. AttxfReplace checks if the attribute is set and able to be replaced. If it is not set, this service call returns the reason code JrAttrUnavail. AttxfCreate checks if the attribute is not set and able to be created (set). If it is already set, BPX1CHR/BPX4CHR will return reason code JrAttrExists.
- The AttxfLset flag controls whether the operation affects a symlink's attributes or the attributes of the symlink's target. When this flag is ON, the operation affects the symlink. When the flag is OFF, the operation affects the symlink's target.
- Changing the general attributes is not supported with fchattr (BPX1FCR, BPX4FCXR) and lchattr (BPX1LCR, BPX4LCR). General attributes include ATTNODELFILES, ATTSHARELIB, ATTNOSHAREAS, ATTAPFAUTH, and ATTPROGCTL, with corresponding extended attributes trusted.apfauth, trusted.sharelib, trusted.progctl, and system.noshareas.
- The SET/REMOVE function affects the attribute information that also exists in BPXYSTAT. The following table provides a correspondence between the information in BPXYATT and BPXYSTAT.

Table 11. Correspondence between BPXYATT and BPXYSTAT			
Xattr name	Existing SET attribute in BPXYATT	Existing GET attribute in BPXYSTAT	Restrictions and comments
trusted.apfauth	attapfauth	ST_APFAUTH	Set and remove functions that are limited according to "general flags" restrictions.
trusted.sharelib	attsharelib	ST_SHARELIB	Set and remove functions that are limited according to "general flags" restrictions.
trusted progctl	attprogctl	ST_PROGCTL	Set and remove functions that are limited according to "general flags" restrictions.
system.noshareas	attnoshareas	ST_NOSHAREAS	Set and remove functions that are limited according to "general flags" restrictions.
system.filefmt	attfilefmt	ST_FILEFMT	None.
system.filetag	attfiletag	ST_FILETAG	None.
system.seclabel	attseclabel	ST_SECLABEL	None.
system.useraudit	attuseraudit	ST_USERAUDIT	None.
system.auditoraudit	attauditoraudit	ST_AUDITORAUDIT	None.
system.auditid	Not an attribute. Equivalent to field ST_AUDITID in BPXYSTAT.	ST_AUDITID	Read-only.
system.dmodelacl	Not an attribute. Equivalent to field.	ST_DMODELACL	Read-only.
system.fmodelacl	Not an attribute. Equivalent to field.	ST_FMODELACL	Read-only.
system.accessacl	Not an attribute. Equivalent to field.	ST_ACCESSACL	Read-only.

Table 11. Correspondence between BPXYATT and BPXYSTAT (continued)

Xattr name	Existing SET attribute in BPXYATT	Existing GET attribute in BPXYSTAT	Restrictions and comments
system.createtime	Not an attribute. Equivalent to field.	ST_CREATETIME	Read-only.
user.*	No equivalent.	No equivalent.	Needs specific PFS support.

## Related services

- [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory ” on page 82](#)

## Characteristics and restrictions

1. The ATTGENVALUE field of BPXYATT cannot be modified with BPX1LCR.
2. The General Attribute fields (set by ATTSETGEN, ATTGENMASK, and ATTGENVALUE fields) are not intended as a general-use programming interface to BPX1LCR.
3. The security label (ATTSECLABELCHG) flag requires RACF SPECIAL authorization and appropriate privileges (see [“Authorization” on page 8](#)). You cannot use it to change an existing security label. It can only be used to set an initial security label on a file.
4. When *Pathname* refers to a symbolic link, any attributes that are requested for change other than owner, times, and security label are ignored.

## Examples

See [“BPX1LCR \(lchattr\) example” on page 1255](#) and [“BPX4LCR \(lchattr\) example” on page 1348](#).

# lchown (BPX1LCO, BPX4LCO) — Change the owner or group of a file, directory, or symbolic link

## Function

The lchown service changes the owner or group (or both) of a file or a directory. The owner is identified by a user ID (UID) and a group ID (GID).

The lchown service is identical to the chown service, except when the Pathname specified is a symbolic link (a pointer to another file or directory). If the Pathname is a symbolic link, the UID and/or the GID of the symbolic link are updated, rather than the UID or GID of the file to which the symbolic link refers. See [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#).

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1LCO):</b>	31-bit
<b>AMODE (BPX4LCO):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts

**Operation****Environment****Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax diagram is as follows:

```
CALL BPX1LCO, (Pathname_length,
               Pathname,
               Owner_UID,
               Group_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4LCO with the same parameters.

**Parameters****Pathname\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the path name of the file for which the owner or group is to be changed.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. The length of this field is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

If the path name specifies a symbolic link file, the lchown service changes the ownership of the symbolic link file itself.

**Owner\_UID**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that contains the new owner UID that is assigned to the file; or the present value or -1, if there is no change. This parameter must be specified.

**Group\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that contains the new owner GID that is assigned to the file; or the present value or -1, if there is no change. This parameter must be specified.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lchown service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lchown service stores the return code. The lchown service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The lchown service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The calling process does not have permission to search some component of the Pathname prefix.
<b>EINVAL</b>	The Owner_UID or Group_ID parameter is incorrect.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters; or a component of the path name is longer than 255 characters.
<b>ENOENT</b>	No file named Pathname was found; or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of the Pathname prefix is not a directory.
<b>EPERM</b>	The calling process does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).
<b>EROFS</b>	Pathname is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lchown service stores the reason code. The lchown service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The lchown service changes the owner UID and owner GID of a file. Only a caller with appropriate privileges (see [“Authorization” on page 8](#)) can change the owner UID of a file.
2. The owner GID of a file can be changed by a caller if the caller has appropriate privileges, or if the caller meets all of these conditions:
  - The effective UID of the caller matches the file's owner UID.
  - The Owner\_UID value that is specified in the change request matches the file's owner UID.
  - The Group\_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner\_UID and Group\_ID must be specified. To change only one of these values, set the other to its present value or to -1.

**Related services**

- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#)
- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1LCO \(lchown\) example” on page 1255](#).

## link (BPX1LNK, BPX4LNK) — Create a link to a file

---

**Function**

The link callable service creates a link to a file. The link is a new name that identifies an existing file. The new name does not replace the old one; it provides an additional way to refer to the file. To rename an existing file, see [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory ” on page 690](#).

**Requirements****Operation****Environment****Authorization**

Supervisor state or problem state, any PSW key

## link (BPX1LNK, BPX4LNK)

Operation	Environment
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1LNK)	31-bit
AMODE (BPX4LNK)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1LNK,(Filename_length,  
              Filename,  
              Link_name_length,  
              Link_name,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4LNK with the same parameters.

## Parameters

### Filename\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Filename of the existing file.

### Filename

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Filename\_length parameter

The name of a field of length Filename\_length that contains the name of the existing file to which a link is to be established.

### Link\_name\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Link\_name.

### Link\_name

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Link\_name\_length parameter

The name of a field that contains the link name by which the file is to be known.

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the link service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the link service stores the return code. The link service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The link service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process did not have appropriate permissions to create the link. Possible reasons include the following reasons: <ul style="list-style-type: none"> <li>• No search permission for a path name component of Filename or Link_name.</li> <li>• No write permission for the directory intended to contain the link.</li> <li>• No permission to access Filename.</li> </ul>
<b>EEXIST</b>	A file, directory, or symbolic link named Link_name already exists. The following reason code can accompany the return code: JRLnkNewPathExists.
<b>EINVAL</b>	The Filename or Link_name is incorrect because it contains a null.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Filename or Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Filename or Link_name.
<b>EMLINK</b>	Filename already has its maximum number of links. The maximum number is LINK_MAX. The value of LINK_MAX can be determined through <a href="#">“pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name” on page 535</a> , or <a href="#">“fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables” on page 233</a> .

Return_code	Explanation
<b>ENAMETOOLONG</b>	Filename or Link_name is longer than 1023 characters or some component of the path name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	A component of the path name that was specified by Filename or Link_name was not found; the file specified by Filename was not found; or one of the two arguments is missing. The following reason code can accompany the return code: JRLnkNoEnt.
<b>ENOSPC</b>	The directory intended to contain the link cannot be extended to contain another entry.
<b>ENOTDIR</b>	A pathname component of one of the arguments is not a directory.
<b>EPERM</b>	Filename is the name of a directory; links to directories are not allowed. The following reason code can accompany the return code: JRLnkDir.
<b>EROFS</b>	Creating the link would require writing on a read_only file system. The following reason code can accompany the return code: JRLnkROFileset.
<b>EXDEV</b>	Filename and Link_name are on different file systems. z/OS UNIX does not support links between file systems. The following reason code can accompany the return code: JRLnkAcrossFilesets.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the link service stores the reason code. The link service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The link service creates a link named Link\_name to an existing file named Filename. This provides an alternative path name for the existing file; the file can be accessed by the old name or the new name. The link can be stored in the same directory as the original file, or in a different directory.
2. If the link is created successfully, the service increments the link count of the file. The link count shows how many links exist for a file. (If the link is not created successfully, the link count is not incremented.)
3. Links are allowed only to files, not to directories.
4. If the link is created successfully, the change time of the linked-to file is updated. The change and modification times of the directory that holds the link are also updated.

**Related services**

- [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor” on page 387](#)
- [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory” on page 690](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)



## Characteristics and restrictions

None.

### Examples

See “BPX1LNK (link) example” on page 1258 and “BPX4LNK (link) example” on page 1351.

## linkat (BPX1LNA, BPX4LNA) – Make a link using at directory relative to a directory file descriptor

---

### Function

The linkat service is identical to link except that it creates a link to an existing file by using a directory file descriptor as the starting location for the path searches for the link and the file. Optional flags indicate how to resolve the path.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1LNA)</b>	31-bit
<b>AMODE (BPX4LNA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1LNA,(Filename_length,
              Filename,
              Link_name_length,
              Link_name,
              FilenameDirFD,
              Link_NameDirFD,
              Flags,
              Return value,
              Return code,
              Reason code)
```

AMODE 64 callers use BPX4LNA with the same parameters.

### Parameters

#### Filename\_length

Supplied parameter.

#### Type

Integer

## linkat (BPX1LNA, BPX4LNA)

### **Length**

Fullword

The name of a fullword that contains the length of the Filename of the existing file.

### **Filename**

Supplied parameter.

### **Type**

Character string

### **Character set**

No restriction

### **Length**

Specified by the Filename\_length parameter

The name of a field of length Filename\_length that contains the name of the existing file to which a link is to be established.

### **Link\_name\_length**

Supplied parameter.

### **Type**

Integer

### **Length**

Fullword

The name of a fullword that contains the length of the Link\_name.

### **Link\_name**

Supplied parameter.

### **Type**

Character string

### **Character set**

No restriction

### **Length**

Specified by the Link\_name\_length parameter

The name of a field that contains the link name by which the file is to be known.

### **FilenameDirFD**

Supplied parameter.

### **Type**

File descriptor.

### **Length**

Fullword.

The name of a fullword that contains a file descriptor for the starting directory to search for the existing file or the special value AT\_FDCWD, which means to use the current working directory as the starting point.

FilenameDirFD is ignored if the path is absolute.

### **Link\_NameDirFD**

Supplied parameter.

### **Type**

File descriptor.

### **Length**

Fullword.

The name of a fullword that contains a file descriptor for the directory to contain the newly created link, or the special value AT\_FDCWD, which means to use the current working directory as the starting point.

Link\_NameDirFD is ignored if the path is absolute.

### Flags

Supplied parameters.

#### Type

Integer.

Can contain zero or one or more of the following values OR'd together.

#### AT\_SYMLINK\_FOLLOW

By default, linkat() does not dereference Filename if it is a symbolic link such as link(). The flag AT\_SYMLINK\_FOLLOW can be specified in flags to cause Filename to be dereferenced if it is a symbolic link.

#### AT\_EMPTY\_PATH

If Filename is an empty string, creates a link to the file referenced by FilenameDirFD, which might have been obtained using the open O\_PATH flag. In this case, FilenameDirFD can refer to any type of file except a directory. This will generally not work if the file has a link count of zero. (Files created with O\_TMPFILE and without O\_EXCL are an exception.)

### Return\_value

Returned parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword in which the linkat service returns 0 if the request is successful or -1 if it is not successful.

### Return\_code

Returned parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword in which the linkat service stores the return code. The linkat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The linkat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process did not have appropriate permissions to create the link. Possible reasons include the following reasons: <ul style="list-style-type: none"> <li>• No search permission for a path name component of Filename or Link_name.</li> <li>• No write permission for the directory intended to contain the link.</li> <li>• No permission to access Filename.</li> </ul>
<b>EBADF</b>	Filename (Link_name) is relative but FilenameDirFD (Link_NameDirFD) is neither AT_FDCWD nor a valid file descriptor.

Return_code	Explanation
<b>EEXIST</b>	A file, directory, or symbolic link named Link_name already exists. The following reason code can accompany the return code: JRLnkNewPathExists.
<b>EINVAL</b>	The Filename or Link_name is incorrect because it contains a null.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Filename or Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Filename or Link_name.
<b>EMLINK</b>	Filename already has its maximum number of links. The maximum number is LINK_MAX. The value of LINK_MAX can be determined through <a href="#">“pathconf (BPX1PCF, BPX4PCF) – Determine configurable path name variables using a path name”</a> on page 535, or <a href="#">“fpathconf (BPX1FPC, BPX4FPC) – Determine configurable path name variables”</a> on page 233.
<b>ENAMETOOLONG</b>	Filename or Link_name is longer than 1023 characters or some component of the path name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	Filename is a relative path name and FilenameDirFD refers to a directory that was deleted, or Link_name is a relative path name and Link_NameDirFD refers to a directory that was deleted.
<b>ENOSPC</b>	The directory intended to contain the link cannot be extended to contain another entry.
<b>ENOTDIR</b>	A path name component of one of the arguments is not a directory.
<b>EPERM</b>	Filename is the name of a directory; links to directories are not allowed. The following reason code can accompany the return code: JRLnkDir.
<b>EROFS</b>	Creating the link would require writing on a read_only file system. The following reason code can accompany the return code: JRLnkROFileset.
<b>EXDEV</b>	Filename and Filename are on different file systems. z/OS UNIX does not support links between file systems. The following reason code can accompany the return code: JRLnkAcrossFilesets.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the linkat service stores the reason code. The link service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The linkat service creates a link named Link\_name to an existing file named Filename. This provides an alternative path name for the existing file; the file can be accessed by the old name or the new name. The link can be stored in the same directory as the original file, or in a different directory.
2. If the link is created successfully, the service increments the link count of the file. The link count shows how many links exist for a file. (If the link is not created successfully, the link count is not incremented.)
3. Links are allowed only to files, not to directories.

4. If the link is created successfully, the change time of the linked-to file is updated. The change and modification times of the directory that holds the link are also updated.

## Related services

- [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file ” on page 383](#)
- [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory ” on page 690](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1LNA \(linkat\) example” on page 1257](#) and [“BPX4LNA \(linkat\) example” on page 1350](#).

# listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients

---

## Function

The listen callable service creates a connection request queue for a server socket to queue incoming connection requests from a client.

Listen is used for connection-oriented sockets only. If a connection request arrives with the backlog queue full, the client may receive an ECONNREFUSED.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1LSN):</b>	31-bit task mode or SRB mode
<b>AMODE (BPX4LSN):</b>	64-bit task mode only
<b>ASC mode:</b>	AR mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1LSN,(Socket_descriptor,
              Backlog,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4LSN with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the listen is to be done.

### Backlog

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the maximum length of the connection queue. For network sockets, if Backlog is greater than SOMAXCONN, this field is set to SOMAXCONN. For AF\_UNIX sockets, there is no maximum value for this field.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword to which the listen service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the listen service stores the return code. The listen service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The listen service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINVAL</b>	An incorrect argument was supplied. The socket is not named (a bind has not been done); or the socket is ready to accept connections (a listen has already been done). The following reason code can accompany the return code: JRListenNotAccepted.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutOfSocketCells.

Return_code	Explanation
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EOPNOTSUPP</b>	The socket descriptor specified a datagram socket. The listen service is valid only for stream sockets. The following reason code can accompany the return code: JRListenNotStream.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the listen service stores the reason code. The listen service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If a bind is not called before the listen request, the listen callable service returns an EINVAL.
2. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

**Characteristics and restrictions**

None.

**Examples**See [“BPX1LSN \(listen\) example” on page 1258](#).

## loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name

---

**Function**

The loadhfs service loads an executable program by path name into the caller's process.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1LOD):</b>	31-bit
<b>AMODE (BPX4LOD):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax diagram is as follows:

```
CALL BPX1LOD, (Filename_length,
               Filename,
               Flags,
               Libpath_length,
               Libpath,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers need an additional parameter, Entry\_point, shown as follows:

```
CALL BPX4LOD, (Filename_length,
               Filename,
               Flags,
               Libpath_length,
               Libpath,
               Entry_point,
               Return_value,
               Return_code,
               Reason_code)
```

**Parameters****Filename\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Filename parameter. The length can be a value in the range 1 to 1023.

**Filename**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Filename\_length parameter

The name of a field that contains the file name of the program that is to be loaded. If the Filename parameter does not contain a slash (/), it is treated as a base name; it should be in one of the directories that are listed in the supplied Libpath parameter. If the Libpath parameter is null, the file must be in the current directory. If the file name is not a base name (that is, it contains at least one slash), the name is used as is; the Libpath parameter is not used to locate the file.

If the file name is a base name, it can be up to 255 characters long.

If the Filename parameter represents a path name, each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.



**Flags**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The Flags parameter is a fullword field that contains option flags that the loadhfs service uses in determining the optional processing to be performed on behalf of the caller. These constants are defined in the BPXYCONS macro.

Constant	Description
<b>Lod_Error_St_ExLink</b>	Indicates that LOAD processing is to be bypassed if the file is an external link or has the sticky bit set on.  If the file is sticky or is an external link, the request fails with return code EPERM (the operation is not permitted) and a reason code of JrExternalLink or JrStickyBit.
<b>Lod_Ignore_Sticky</b>	Indicates that the sticky bit for a file is to be ignored. If the file is sticky, it is loaded from the z/OS UNIX file system.

If both Lod\_Ignore\_Sticky and Lod\_Error\_St\_ExLink are specified, the Lod\_Ignore\_Sticky option is honored, and Lod\_Error\_St\_ExLink is ignored.

**Libpath\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the library path parameter. If a value of zero is specified, the library path parameter is ignored.

**Libpath**

Supplied parameter

**Type:**

Structure

**Length:**

Specified by the Libpath\_length parameter

The name of a field that contains the library path to be searched to determine the fully qualified path name of the file that is specified. The library path can contain a series of path names that are separated by colons. The path names in the list are searched one at a time until the specified file name is located. If the list of path names begins or ends with a colon, the working directory of the calling process is used to locate the file. Each path name in the list can have a maximum length of 1021 bytes.

The following is an example of a valid library path:

- /usr1/bin:/grp1/bin:/bin

**Entry\_point**

Returned parameter (BPX4LOD only)

**Type:**

Structure

**Length:**

Doubleword

The name of a field that contains the entry point.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the loadhfs service returns -1 if it is not successful. If it is successful, the loadhfs service returns the entry point address of the program that was loaded into storage. If the loaded program is an AMODE 31 program, the high-order bit of the return value is turned on. For this reason, applications that test for a failure condition must explicitly check for a -1 return value. Checking for a value of less than zero will not produce the desired results.

For AMODE 64 programs, if the return value is 0, the entry point address of the loaded program is returned in the Entry\_point parameter.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the loadhfs service stores the return code. The loadhfs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The loadhfs service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The caller does not have appropriate permissions to run the specified file. It might lack permission to search a directory that is named in the Pathname parameter; it might lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type.
<b>EAGAIN</b>	The file changed during load processing (JrFileChangeDuringLoad).
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Filename parameter. This error is issued if more than 24 symbolic links are detected in the resolution of Filename.
<b>EMVSERR</b>	One of the following situations occurred: <ul style="list-style-type: none"> <li>• An error occurred while z/OS UNIX program (JrMVSLoadFailure or JrMVSPgmNotFound) was being loaded.</li> <li>• An error occurred while the caller's environment against the authorization of the file (JrNoListAuthPgmPath, JrNoListPgmCntlPath, JrProgCntl, JrAuthCaller) was being checked.</li> <li>• The caller attempted to load an RMODE 64 program while running AMODE 31 (JrRM64AM31).</li> </ul>
<b>ENAMETOOLONG</b>	Filename is longer than 1023 characters; or some component of the file name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file name was specified, or one or more of the components of the specified Filename parameter were not found.
<b>ENOEXEC</b>	The specified file has execute permission, but it is not in the proper format to be a process image file.

Return_code	Explanation
<b>ENOMEM</b>	The file that is to be loaded requires more memory than is permitted by the hardware or the operating system.
<b>ENOTDIR</b>	A directory component of Filename is not a directory.
<b>EINVAL</b>	An invalid parameter value was specified. The invalid parameter might be one of the following: Filename_length.
<b>EPERM</b>	The operation is not permitted. The Flags parameter was set to Lod_Error_St_ExLink, and either the file is an external link (JrExternalLink), or it has the sticky bit set on (JrStickyBit).

**Note:** In addition to the return codes listed here, the loadhfs service can return additional errors for other failures that can occur on a stat or an open syscall.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the loadhfs service stores the reason code. The loadhfs service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### Usage notes for loadhfs

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes loadhfs to load a z/OS UNIX executable program file, the program file and any symbolic links that comprise the path name to the file must have the proper attributes to allow the invocation. The z/OS UNIX executable file must be installed with an owning UID of 0 or with an owning UID equal to that of the calling set-user-ID program or have the program control extended attribute. The same is true for any symbolic link that comprises that path name to the file or any external symbolic link, except that, the program control extended attribute cannot be used because links do not have extended attributes. Additionally, a file or link found in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Not following this setup causes the caller of loadhfs to end abnormally with a EC6-xxxxE04B abend.
2. A prior loaded copy of a z/OS UNIX program is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS LOAD service, with the following exceptions:
  - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
  - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the z/OS UNIX program found is in storage that is modifiable by the caller, the prior copy is not reused.
3. If the specified file name represents an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is only used if the name is 8 characters or less, otherwise the caller receives an error from the loadhfs service. For a sticky bit program, the file name is used if it is 8 characters or less. If the file name is greater than 8 characters, or the MVS program is not found, the program is loaded from the z/OS UNIX file system.
4. When it is running from a pthread\_created thread (pthread), the specified file is loaded into storage and associated with the Initial Pthread Creating Task (IPT). This allows the program to be shared across multiple threads, without the problem of its disappearing unexpectedly when a thread terminates.
5. When the calling process is being debugged via the ptrace service, the following applies:

- Programs that are loaded using this service are loaded into storage that is modifiable by the caller of the loadhfs service.
  - A call to this service generates a WastStopFlagLoad Ptrace event to the debugger process.
6. Because this service does not cause the specified program to be executed, the set-user-ID and set-group-ID flags have no impact on the process.
  7. Because the z/OS UNIX file system is not an authorized library, the following restrictions apply:
    - Loading a program from the z/OS UNIX file system causes the program environment to become uncontrolled unless the executable file has the program control attribute turned on (ST\_PROGCTL). Having the program control attribute on prevents future invocations of authorized programs like PADS programs. In addition, PADS programs should not attempt to load programs from the z/OS UNIX file system; the z/OS UNIX file system is considered an unauthorized library and can potentially be modified by users that do not have the same level of authorization as the PADS program.
    - System key, supervisor state, and APF-authorized callers should not attempt to load a program from the z/OS UNIX file system, unless the executable file has the APF attribute turned on.
  8. If a program that is loaded into storage with this service is not deleted from storage, the program remains in storage until the calling task terminates, if it is not a pthread. If the caller is a pthread, the program remains in storage until the Initial Pthread Creating Task (IPT) terminates.
  9. The AUTHPGMLIST environment variable works with this system call. The environment variable specifies a list of sanctioned directories or authorized program names. If activated, an additional level of security checking will be performed to ensure that the program being loaded is coming from an authorized directory in the z/OS UNIX file system or is an authorized MVS program name. For more information about the sanction list, see [Using sanction lists in z/OS UNIX System Services Planning](#).

The following usage notes apply for shared library programs:

10. Executables that have the ST\_SHARELIB extended attribute turned on are considered system shared library programs. System shared library programs are the most optimal way to share large executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single page table (similar to LPA). The storage that is used in the user address space to establish the mapping to the shared library region is from the high end of private storage; it does not interfere with the virtual storage used by the application program.
11. If the program to be loaded is determined to be a shared library program (that is, if the ST\_SHARELIB extended attribute is on), the loadhfs service queries the shared library region to determine if the target program is there.

When a shared library program is loaded anew into the shared region or reloaded from the shared region, the program is mapped from the shared region into the private area of the calling address space. It is important to note that, because the program is not actually reloaded from DASD into the private area of each calling address space, but only remapped from the shared region, shared library programs are more efficient in their utilization of system resources than normal private area programs. For this reason, programs that are to be shared across several address spaces in the system are good candidates for identification as shared library programs.

If a target program is not in the shared library region and cannot be loaded into the region because of its attributes, the program is treated like a private area program and is loaded into the caller's private area storage.

Additionally, if the calling address space cannot accommodate the target address for the shared library program, the program is treated like a private area program.

12. In order for a program to be honored as a shared library program, certain conditions must be met:
  - The program must be a z/OS UNIX program module; MVS library modules cannot be loaded into the shared region.
  - A sticky bit program that is found in the MVS search order is not honored as a shared library program.

- The program cannot be a multiple-segment (split RMODE) load module; multiple-segment load modules are not supported in the shared library region.
  - The program must have read "other" permission and be link-edited as REENTRANT.
13. A shared library program can reside in a file system that was mounted with the NOSETUID operand.
14. The caller must be running AMODE 64 when loading an RMODE64 program.

## Related services

- [“deletehfs \(BPX1DEL, BPX4DEL\) — Delete a program from storage ” on page 148](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1LOD \(loadhfs\) example” on page 1257](#).

# loadhfs extended (BPX1LDX, BPX4LDX) — Direct the loading of an executable into storage

---

## Function

The loadhfs extended service loads an executable program by path name into the caller process. This service provides all the functions of [“loadhfs \(BPX1LOD, BPX4LOD\) — Load a program into storage by path name” on page 393](#) and also allows authorized users to load an executable program into common storage.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key unless the Lod_Directed flag is specified. When this flag is specified, the caller must be APF-authorized, PSW Key 0-7, or Supervisor State.
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1LDX):</b>	31-bit
<b>AMODE (BPX4LDX):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1LDX, (Filename_length,
               Filename,
               Flags,
               Libpath_length,
```

```

Libpath,
Return_value,
Return_code,
Reason_code)

```

AMODE 64 callers need an additional parameter, Entry\_point, as shown in the following syntax format:

```

CALL BPX4LDX,(Filename_length,
               Filename,
               Flags,
               Libpath_length,
               Libpath,
               Entry_point,
               Return_value,
               Return_code,
               Reason_code)

```

## Parameters

### Filename\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the Filename parameter. The length can be a value in the range 1 - 1023.

### Filename

Supplied parameter

#### Type:

Character string

#### Character set:

No restriction

#### Length:

Specified by the Filename\_length parameter

The name of a field that contains the file name of the program that is to be loaded. If the Filename parameter does not contain a slash (/), it is treated as a base name. This parameter must be in one of the directories listed in the supplied Libpath parameter. If the Libpath parameter is null, the file must be in the current directory. If the file name is not a base name (that is, it contains at least one slash), the name is used as is; the Libpath parameter is not used to locate the file.

If the file name is a base name, it can be up to 255 characters long.

If the Filename parameter represents a path name, each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending null character.

### Flags

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The Flags parameter is a fullword field. The first three bytes contain option flags. The last byte can be data as defined by an option flag. These constants are defined in the BPXYCONS macro.

**Constant****Description****Lod\_Directed**

Indicates that the target program is to be loaded into the supplied storage subpool. When this option flag is specified, the storage subpool is supplied as the last byte of the FLAGS parameter. This flag is only supported for authorized system callers (APF-authorized or system key or supervisor state). Unauthorized callers specifying this flag receive a EPERM error return code. When this flag is specified, it is the responsibility of the caller to free the program storage. Only subpool 241 is supported; any other subpool specified results in an EINVAL error return code. The storage that is obtained for the target program is key 0 storage. Lod\_Directed takes precedence over Lod\_Ignore\_Sticky, which in turn takes precedence over Lod\_Error\_St\_ExLink.

For the directed load to common of an RMODE 64 program, the last byte of the FLAGS parameter is ignored. The target program is loaded into above the bar common storage.

**Lod\_Error\_St\_ExLink**

Indicates that LOAD processing is to be bypassed if the file is an external link or has the sticky bit set on.

If the file has the sticky bit set or is an external link, the request fails with return code EPERM (the operation is not permitted) and a reason code of JrExternalLink or JrStickyBit.

**Lod\_Ignore\_Sticky**

Indicates that the sticky bit for a file is to be ignored. If the file is sticky, it is loaded from the z/OS UNIX file system.

If both Lod\_Ignore\_Sticky and Lod\_Error\_St\_ExLink are specified, the Lod\_Ignore\_Sticky option is honored, and Lod\_Error\_St\_ExLink is ignored.

**Libpath\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the library path parameter. If a value of zero is specified, the library path parameter is ignored.

**Libpath**

Supplied parameter

**Type:**

Structure

**Length:**

Specified by the Libpath\_length parameter

The name of a field that contains the library path to be searched to determine the fully qualified path name of the file that is specified. The library path can contain a series of path names that are separated by colons. The path names in the list are searched one at a time until the specified file name is located. If the list of path names begins or ends with a colon, the working directory of the calling process is used to locate the file. Each path name in the list can have a maximum length of 1021 bytes.

The following is an example of a valid library path:

## loadhfs extended (BPX1LDX, BPX4LDX)

- /usr1/bin:/grp1/bin:/bin

### Entry\_point

Returned parameter (BPX4LDX only)

#### Type:

Structure

#### Length:

Doubleword

The name of a field in which either an entry point address or the address of a structure is returned. If the Lod\_Directed flag is specified, this service returns the address of a 24-byte structure that contains the length of the loaded program storage, followed by the start address of the loaded program, followed by the entry point address of the loaded program. The returned structure is mapped in the BPXYCONS macro.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The return value for this service is as follows:

- For an AMODE(31) caller, the name of a fullword in which the loadhfs extended service returns -1 if it is not successful. If it is successful, the loadhfs extended service returns the entry point address of the program that was loaded into storage, unless the Lod\_Directed flag is specified. If the Lod\_Directed flag is specified, this service returns the address of a 24-byte structure that contains the length of the loaded program storage, followed by the start address of the loaded program, followed by the entry point address of the loaded program. If the loaded program is an AMODE(31) program, the high-order bit of the entry point address is ON. The returned structure is mapped in the BPXYCONS macro
- For an AMODE(64) caller, the Return\_value is returned as either 0 if successful or -1 if not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the loadhfs extended service stores the return code. The loadhfs extended service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The directed loadhfs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EACCES	The caller does not have appropriate permissions to run the specified file. It might lack permission to search a directory named in the Pathname parameter; it might lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type.
EAGAIN	The file changed during load processing (JrFileChangeDuringLoad).



Return_code	Explanation
<b>EINVAL</b>	An invalid parameter value was specified. The invalid parameter can be Filename_length, or FLAGS. If FLAGS is incorrect, a reason code of either JrOptionFlagsErr (unsupported FLAGS parameter value), or JrLodDirectedSubpoolError (unsupported value for the directed loadhfs subpool passed in the FLAGS parameter).
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Filename parameter. This error is issued if more than 24 symbolic links are detected in the resolution of Filename.
<b>EMVSERR</b>	One of the following situations occurred: <ul style="list-style-type: none"> <li>• An error occurred while a z/OS UNIX program (JrMVSLoadFailure or JrMVSPgmNotFound) was being loaded.</li> <li>• An error occurred when the caller's environment against the authorization of the file (JrNoListAuthPgmPath, JrNoListPgmCntlPath, JrProgCntl, JrAuthCaller) was checked.</li> <li>• The caller attempted to load an RMODE 64 program while running AMOCE 31 (JrRM64AM31).</li> </ul>
<b>ENAMETOOLONG</b>	The Filename parameter is longer than 1023 characters; or some component of the file name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file name was specified, or one or more of the components of the specified Filename parameter were not found.
<b>ENOEXEC</b>	The specified file has execute permission, but it is not in the proper format to be a process image file.
<b>ENOMEM</b>	The file that is to be loaded requires more memory than is permitted by the hardware or the operating system, or a storage request failed for the directed load target (JrLodDirectedNoStorage).
<b>ENOTDIR</b>	A directory component of the Filename parameter is not a directory.
<b>EPERM</b>	The operation is not permitted. The Flags parameter was set to Lod_Error_St_ExLink. If the file has the sticky bit set or is an external link, the request fails with reason code of JrStickyBit or JrExternalLink, respectively. Or an unauthorized caller specified the Lod_Directed option flag (JrLodDirectedAuthErr).

In addition to the return codes listed here, the loadhfs extended service can return additional errors for other failures that can occur on a stat or an open syscall.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the loadhfs extended service stores the reason code. The loadhfs extended service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

## Usage notes for loadhfs

Note that usage notes 1-9 do not apply if you specify the Lod\_Directed flag.

1. A prior loaded copy of a z/OS UNIX program is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS LOAD service, with the following exceptions:
  - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
  - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the zFS program found is in storage that is modifiable by the caller, the prior copy is not reused.
2. If the specified file name represents an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is only used if the name is eight characters or less, otherwise the caller receives an error from the loadhfs service. For a sticky bit program, the file name is used if it is eight characters or less. If the file name is greater than eight characters, or the MVS program is not found, the program is loaded from the z/OS UNIX file system.
3. When it is running from a pthread\_created thread (pthread), the specified file is loaded into storage and associated with the Initial Pthread Creating Task (IPT). This allows the program to be shared across multiple threads, without the problem of its disappearing unexpectedly when a thread terminates.
4. When the calling process is being debugged via the ptrace service, the following applies:
  - Programs that are loaded using this service are loaded into storage that is modifiable by the caller of the loadhfs service.
  - A call to this service generates a WastStopFlagLoad Ptrace event to the debugger process.
5. Because this service does not cause the specified program to be executed, the set-user-ID and set-group-ID flags have no impact on the process. These flags have meaning only for an execed or spawned program.
6. Because the z/OS UNIX file system is not an authorized library, the following restrictions apply:
  - Loading a program from the z/OS UNIX file system causes the program environment to become uncontrolled unless the executable file has the program control attribute turned on (ST\_PROGCTL). Not having the program control attribute on prevents future invocations of authorized programs like PADS programs. In addition, PADS programs should not attempt to load programs from the z/OS UNIX file system; the z/OS UNIX file system is considered an unauthorized library and can potentially be modified by users that do not have the same level of authorization as the PADS program.
  - System key, supervisor state and APF-authorized callers receive an EMVSERR with reason code JrAuthCaller if the caller attempts to load a program from the z/OS UNIX file system, unless the executable file has the APF attribute turned on.
7. If a program that is loaded into storage with this service is not deleted from storage, the program remains in storage until the calling task terminates, if it is not a pthread. If the caller is a pthread, the program remains in storage until the Initial Pthread Creating Task (IPT) terminates.
8. The AUTHPGMLIST system parameter applies to this system call. AUTHPGMLIST specifies a z/OS UNIX file that contains a list of sanctioned directories or authorized program names. If activated, an additional level of security checking will be performed to ensure that the program being loaded is coming from an authorized directory in the z/OS UNIX file system or is an authorized MVS program name. For details about the sanction list, see [Using sanction lists in z/OS UNIX System Services Planning](#).
9. The following apply to shared program libraries:
  - Executables that have the ST\_SHARELIB extended attribute turned on are considered system shared library programs. System shared library programs are the most optimal way to share large executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single page table (similar to LPA). The storage used in the user address space to establish the mapping to the shared library region is from the high end of private storage.

- If the program to be loaded is determined to be a shared library program (that is, if the ST\_SHARELIB extended attribute is on), the loadhfs service queries the shared library region to determine if the target program is there.

When a shared library program is loaded anew into the shared region or reloaded from the shared region, the program is mapped from the shared region into the private area of the calling address space. It is important to note that, because the program is not actually reloaded from DASD into the private area of each calling address space, but only remapped from the shared region, shared library programs are more efficient in their utilization of system resources than normal private area programs. For this reason, programs that are to be shared across several address spaces in the system are good candidates for identification as shared library programs.

If a target program is not in the shared library region and cannot be loaded into the region because of its attributes, the program is treated like a private area program and is loaded into the caller's private area storage.

Additionally, if the calling address space cannot accommodate the target address for the shared library program, the program is treated like a private area program.

- In order for a program to be honored as a shared library program, certain conditions must be met:
  - The program must be a z/OS UNIX program module; MVS library modules cannot be loaded into the shared region.
  - A sticky bit program that is found in the MVS search order is not honored as a shared library program.
  - The program cannot be a multiple-segment (split RMODE) load module; multiple-segment load modules are not supported in the shared library region.
  - The program must have read "other" permission and be link-edited as REENTRANT.
- A shared library program can reside in a file system that was mounted with the NOSETUID operand.

#### 10. When the Lod\_Directed flag is specified:

- It is the responsibility of the caller to manage the storage that is associated with the loaded program. When Lod\_Directed is specified, deletehfs cannot be used to remove the executable from storage. The executable will stay in storage until freed. The storage can be freed using the returned storage length and program start address.
- It is the responsibility of the caller to use the CSVDYLPAA ADD BYADDR(YES) service to create a CDE in order to provide serviceability information for the loaded program. Without this, serviceability functions, such as SLIP LPAMOD and IPCS WHERE, are not available for the loaded program.
- The caller must save a copy of the returned program information after each call. The returned data structure is reused for each syscall by a given task. The returned program information structure is cleared if the call is made and an error occurs.
- A program loaded with the Lod\_Directed flag cannot be debugged using Ptrace debug mode.
- The shared library program attribute, st\_Sharelib, is ignored.
- The sticky bit for a file is ignored whether or not Lod\_Ignore\_Sticky is specified.
- If the file is an external link, the request will fail with return code of EPERM (the operation is not permitted) and a reason code of JrExternalLink whether or not Lod\_Error\_St\_ExLink is specified.

#### 11. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes loadhfs extended to load a z/OS UNIX executable program file, the program file and any symbolic links that comprise the path name to the file must have the proper attributes to allow the invocation. The z/OS UNIX executable file must be installed with an owning UID of 0 or with an owning UID equal to that of the calling set-user-ID program or have the program control extended attribute. The same is true for any symbolic link that comprises that path name to the file or any external symbolic link, except that, the program control extended attribute cannot be used because links do not have extended attributes. Additionally, a file or link found in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Not following this setup causes the caller of loadhfs to end abnormally with a EC6-xxxxE04B abend.

12. The caller must be running AMODE 64 when loading an RMODE 64 program.

## Related services

None.

## Characteristics and restrictions

None.

## Examples

See “BPX1LDX (loadhfs extended) example” on page 1255.

# lseek (BPX1LSK, BPX4LSK) – Change a file's offset

---

## Function

The lseek callable services changes the file offset of a file to a new position. The file offset is the position in a file from which data is next read, or to which data is next written.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1LSK):</b>	31-bit
<b>AMODE (BPX4LSK):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1LSK, (File_descriptor,
               Offset,
               Reference_point,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4LSK with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor for the file whose file offset is to be changed. The file descriptor is returned from [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511.

**Offset**

Parameter supplied and returned

**Type:**

Integer

**Length:**

Doubleword

The name of a doubleword that contains a signed number. The numeric part of the value is the amount (number of bytes) by which you want to change the offset. The sign indicates whether you want the offset to be moved forward or backward in the file.

This field is a doubleword, to accommodate large files. For normal processing with a singleword value, propagate the sign bit through the second word, so that the final doubleword value has a valid sign.

On successful completion, this field returns the new file offset.

**Reference\_point**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains a value that represents an option. Reference\_point indicates the point from which the offset is calculated. These values are mapped by the BPXYSEEK macro. For information on the contents of the macro, see [“BPXYSEEK — Constants for lseek”](#) on page 1144.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lseek service returns 0 if the request is successful, or -1 if it is not successful. Offset returns the new file offset if the request is successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lseek service stores the return code. The lseek service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The lseek service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter does not specify a valid, open file.

Return_code	Explanation
<b>EINVAL</b>	The Reference_point parameter contained something other than one of the three options; or the combination of the Offset and Reference_point parameters would have placed the file offset before the beginning of the file. The following reason codes can accompany the return code: JRLskOffsetIsInvalid, JRLskWhenceIsInvalid.
<b>ESPIPE</b>	The File_descriptor refers to a pipe, a FIFO special file, or a socket. The following reason code can accompany the return code: JRLskOnPipe.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lseek service stores the reason code. The lseek service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. The Offset parameter gives the length and direction of the offset change. Reference\_point parameter states where the change is to start. For example, assume that a file is 2000 bytes long, and that the current file offset is 1000:

Offset specified	Reference point	New file offset
<b>80</b>	SEEK_CUR	1080
<b>1200</b>	SEEK_SET	1200
<b>-80</b>	SEEK_END	1920
<b>132</b>	SEEK_END	2132

2. The file offset can be moved beyond the end of the file. If data is written at the new file offset, there is a gap between the old end of the file and the start of the new data. A request to read data from anywhere within that gap completes successfully, and returns bytes with the value of zero in the buffer and the actual number of bytes read.

Seeking itself, however, does not extend the file. Only if data is written at the new offset does the length of the file change.

**Related services**

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

**Characteristics and restrictions**

When automatic conversion is enabled to ALL (that is, Unicode conversion is in effect), a lseek operation for a file containing multibyte characters can cause a subsequent read (BPX1RED/BPX4RED) or write (BPX1WRT/BPX4WRT) operation to fail due to the following cases:

1. The cursor jumps to another code page in a file tagged with a CCSID that has multiple code pages. The subsequent read will fail. However, using lseek to position the cursor to the beginning of the file is acceptable. The code page will be reset to the beginning default defined for the CCSID.
2. The previous write operation caused LFS to internally cache an incomplete multibyte character which, as a result of the lseek, is no longer convertible. The subsequent read or write will fail.
3. The cursor jumps to a position which is not on a character boundary. The subsequent read will fail.

### Examples

See [“BPX1LSK \(lseek\) example” on page 1258](#).

## lstat (BPX1LST, BPX4LST) – Get status information about a file or symbolic link by path name

---

### Function

The lstat callable service obtains status information about a file. The lstat service is identical to the stat service, except when the path name specified is a symbolic link (a pointer to another file or directory). In this case, the status information that is returned relates to the symbolic link, rather than to the file to which the symbolic link refers. The stat service is explained in [“stat \(BPX1STA, BPX4STA\) – Get status information about a file by path name” on page 893](#).

For the corresponding service using a file descriptor, see [“fstat \(BPX1FST, BPX4FST\) – Get status information about a file by descriptor” on page 238](#).

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1LST)</b>	31-bit
<b>AMODE (BPX4LST)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1LST, (Pathname_length,
               Pathname,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4LST with the same parameters.

**Parameters****Pathname\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of Pathname.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of an area of length Pathname\_length that contains the path name of the file for which you want to obtain status. The Pathname can be a path name to a file, a link name to a file (as created by [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file ” on page 383](#)), or a symbolic link name (as created by [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)).

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Status\_area\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the area to which the service returns Status\_area. To determine the value of Status\_area\_length, use macro BPXYSTAT; see [“BPXYSTAT — Map the response structure for stat ” on page 1162](#).

**Status\_area**

Parameter supplied and returned

**Type:**

Structure

**Length:**

Length of BPXYSTAT macro

The name of an area of length Status\_area\_length to which the service returns the status information for the file. Status\_area is mapped by the BPXYSTAT macro; see [“BPXYSTAT — Map the response structure for stat ” on page 1162](#).

**Return\_value**

Returned parameter

**Type:**

Integer



**Length:**

Fullword

The name of a fullword in which the lstat service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lstat service stores the return code. The lstat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The lstat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process does not have permission to search some component of the Pathname prefix.
<b>EINVAL</b>	Parameter error; for example, a zero-length buffer. The following reason code can accompany the return code: JRBuffTooSmall.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters; or some component of the path name is longer than 255 characters. This could happen if a symbolic link was encountered during the resolution of Pathname, and the substituted string was longer than 1023 characters.
<b>ENOENT</b>	No file named Pathname was found, or Pathname was not specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	A component of the Pathname prefix is not a directory.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the lstat service stores the reason code. The lstat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. All time fields in Status\_area are in POSIX format.
2. The File Mode field in Status\_area is mapped by BPXYMODE; see “[BPXYMODE — Map the mode constants](#)” on page 1106. For information on the values for file type, see “[BPXYFTYP — File type definitions](#)” on page 1078.
3. If no security label (SECLABEL) exists for the file, the security label field in the Status\_area contains binary zeros.

**Related services**

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fpathconf \(BPX1FPC, BPX4FPC\) — Determine configurable path name variables” on page 233](#)
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#)
- [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor ” on page 241](#)
- [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file ” on page 383](#)
- [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor ” on page 387](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe” on page 551](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“utime \(BPX1UTI, BPX4UTI\) — Set file access and modification times ” on page 981](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

**Characteristics and restrictions**

To obtain information about a file, you need not have permissions for the file itself; however, you must have search permission for all of the directory components of Pathname.

**Examples**

See [“BPX1LST \(lstat\) example” on page 1258](#).

## **\_\_map\_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area**

---

**Function**

The `__map_init` callable service creates a mapped megabyte area in the private area of the calling address space to hold a fixed number of the application's data blocks. This map area is divided into map blocks, each of which is a view onto a data block that is maintained in the kernel data space. The application can set the number of map blocks contained in the map area and the size, in megabytes, of each map block.

Once it has created the map area with the `__map_init` service, an application can use the `__map_service` (BPX1MMS, BPX4MMS) callable service to connect and disconnect blocks of storage in the map area.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Problem program or supervisor state, PSW key 8
<b>Dispatchable unit mode</b>	Task

Operation	Environment
Cross memory mode	PASN = HASN
AMODE (BPX1MMI)	31-bit
AMODE (BPX4MMI)	64-bit
ASC mode	Primary address space control (ASC) mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1MMI (FunctionCode,
              ParmListPtr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MMI with the same parameters. ParmListPtr is a doubleword pointer field.

Parameters

FunctionCode

Supplied parameter

Type:

Integer

Length:

Fullword

The name of a fullword that contains a value indicating the type of map function the caller is requesting. The following is the only supported value:

Constant	Description
MMG_INIT	Create mapped megabyte area

This constant is defined in the BPXYMMG macro. See [“BPXYMMG — Map interface for \\_map\\_init and \\_map\\_service ” on page 1101.](#)

ParmListPtr

Supplied parameter

Type:

Pointer

Length:

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the parameter list. See [“BPXYMMG — Map interface for \\_map\\_init and \\_map\\_service ” on page 1101](#) for the mapping of the parameter list.

Return\_value

Returned parameter

Type:

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_map\_init service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_map\_init service stores the return code. The \_\_map\_init service stores a return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_map\_init service may return one of the following values in the Return\_code parameter:

<b>Return code</b>	<b>Explanation</b>
<b>EEXIST</b>	An attempt was made to create more than one map area for the process (JRMappedAlreadyActive).
<b>ENOMEM</b>	A request to initialize a map area failed for one of the following reasons: <ul style="list-style-type: none"> <li>• There was insufficient storage in the caller's address space to obtain the map area needed to contain the map blocks (JRNoUserStorage).</li> <li>• All or part of the area defined by the address that was provided by the caller in MMG_AREAADDR was already allocated (JRStorageNotAvail).</li> </ul>
<b>EPERM</b>	One of the following errors occurred: <ul style="list-style-type: none"> <li>• The caller is not permitted to the BPX.MAP resource in the FACILITY class. Superuser status (UID=0) is not sufficient (JRNotAuthMAP).</li> <li>• The BPX.MAP resource in the FACILITY class is not defined, and the user is not a superuser (JRNotSuper).</li> </ul>
<b>EMVSSAF2ERR</b>	An error occurred in the security product.
<b>EINVAL</b>	One of the following errors occurred: <ul style="list-style-type: none"> <li>• The FunctionCode parameter contains a value that does not represent a supported function (JRMapBadFunction).</li> <li>• The number of blocks specified ( _MMG_NUMBLKS) was either negative or zero (JRNegativeValueInvalid).</li> <li>• The number of megabytes per block specified ( _MMG_MEGSPERBLK) was either negative or zero (JRNegativeValueInvalid).</li> <li>• A reserved field contains nonzero data (JRReservedValueInvalid).</li> <li>• The request specified a map address ( _MMG_AREAADDR) that was not above the line, or that was not on a megabyte boundary (JRBadAddress).</li> </ul>
<b>EFAULT</b>	An argument of this service contained an address that was not accessible to the caller (JRMapBadStorage).
<b>EMVSERR</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• There was an unexpected error (JRMapUnexpectedErr).</li> <li>• An attempt to process the new map area failed in RSM (JRMapRSM).</li> </ul>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_map\_init service stores the reason code. The \_\_map\_init service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The \_\_map\_init and \_\_map\_service callable services allow applications to manage an unlimited number of data blocks, each of which can hold some number of megabytes of data. They provide a fast way to connect up to persistent memory for applications that need more shared memory than will fit in the address space.
2. It is intended that an application will call the \_\_map\_init service once to create the map area. The map area should be large enough for the biggest expected usage.
3. A process may have one, and only one, map active at a given time. There is currently no way to terminate a map area once it has been established without terminating the establishing process.
4. At any point in time, an application can view as many data blocks as were specified at initialization of the map area, and it can have many times this number of data blocks defined and residing in kernel data spaces.
5. The map area may be shared among one or more processes. Sharing may only be between a parent and any children that were created after the parent created the map area with a call to the \_\_map\_init service. Children that were created before the call do not have access to the map area, nor can they gain access to it through any service.
6. A map area is not propagated across a spawn or preserved across an exec. Unlike most attributes on fork, the map area that is inherited by a child is empty; none of the map blocks are connected to data blocks, regardless of how many data blocks are currently connected to the parent's map area.
7. A map area persists until the process that created it terminates. Once that process terminates, all map activity against the data blocks is shut down. Currently connected blocks may continue to be used until they are disconnected. New blocks cannot be created, nor can a process connect to an existing data block. Once all data blocks have been disconnected by all processes, the map area is ended. A process that has been detached from a map area by disconnecting from all data blocks may create a new map area.
8. Each process that is sharing a map (parent, child, or grandchild) gets a map area that is located at the same virtual storage address as the map originator and that consists of map blocks that are the same size and number as those of the originator. Each process that is sharing a map manages its own map area in terms of the data blocks that are connected, and each process determines which data block is viewed through which map area block.
9. The initial process forks worker processes, which inherit the map area at the same virtual address. Because the map area is at the same virtual address, storage blocks can be connected to the same block in map areas of different worker processes, and pointers can be used to point to data in this and other blocks. (This assumes that they are always connected at the same location in the map area.)
10. As worker processes perform their tasks, they can request that new blocks of storage be created in the map area. Each block has a token associated with it, which allows other worker processes to connect to the same block. In this respect, the map area acts like shared memory.
11. The worker processes can connect as many blocks to their map area as will fit.

12. When the worker process has no further need for a data block, it can disconnect it from the map area. Following a delete request for a block, the block is actually freed when the last worker process disconnects from it.
13. When a worker process has finished using a data block, the storage can be freed. The data is actually freed when the last worker process disconnects from that block.
14. Using the \_\_map\_init and \_\_map\_services, an application could create multiple gigabytes of storage, of which only certain blocks are mapped into the worker processes at a given time.
15. There is no explicit call to delete the map area.

## Related services

- [“\\_\\_map\\_service \(BPX1MMS, BPX4MMS\) — Mapped megabyte area services” on page 416](#)

## Characteristics and restrictions

Users of \_\_map\_service can create and manage a tremendous amount of data, causing the kernel to consume a large amount of system resources. To prevent abuse of such power, the \_\_map\_init service requires that the user be permitted to the BPX.MAP resource in the FACILITY class. (The \_\_map\_service callable service does not check for authority to BPX.MAP, because it does not perform any functions without first completing a \_\_map\_init request.)

## Examples

See [“BPX1MMI \(\\_\\_map\\_init\) example” on page 1261](#).

# **\_\_map\_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services**

---

## Function

The \_\_map\_service callable service performs the following operations on one or more data blocks in a memory map area that was created by the \_\_map\_init service:

- Creates a new data block.
- Connects to an existing data block.
- Disconnects from a data block.
- Frees the backing storage for a data block.
- Changes the read or write permission for a data block.

Before an application can use this service, it must invoke the \_\_map\_init callable service to create a mapped megabyte area to hold its data blocks. See [“\\_\\_map\\_init \(BPX1MMI, BPX4MMI\) — Create a mapped megabyte area” on page 412](#).

## Requirements

Operation	Environment
<b>Authorization</b>	Problem program or supervisor state, PSW key 8
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MMS)</b>	31-bit
<b>AMODE (BPX4MMS)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode

<b>Operation</b>	<b>Environment</b>
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MMS (FunctionCode,
              ParmListPtr,
              ArrayCount,
              BlockToken,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MMS with the same parameters. ParmListPtr is a doubleword pointer field.

## Parameters

### FunctionCode

Supplied parameter.

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value indicating the type of map function the caller is requesting. The following constant is the only supported value: MMG\_SERVICE. It performs one or more operations on map blocks:

- Activates a new data block (MAP\_NEWBLOCK).
- Connects to a data block (MAP\_CONN).
- Disconnects from a data block (MAP\_DISCONN).
- Frees the backing storage for a data block (MAP\_FREE).
- Changes the read or write permissions for a data block (MAP\_CNTL).

These constants are defined in the BPXYMMG macro. See [BPXYMMG](#).

### ParmListPtr

Supplied parameter.

#### Type:

Pointer

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the parameter list for the specified function. See [BPXYMMG](#) for the mapping of the parameter lists.

### ArrayCount

Supplied parameter.

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the number of entries in the array that is contained in the parameter list that is provided by ParmListPtr. The value specified in the ArrayCount parameter must be greater than or equal to 1 and less than or equal to 1000.

**BlockToken**

An 8-byte token value representing the mapped block. The token is returned in the BPXYMMG data area in field \_MMG\_TOKEN for function code \_newblock. This token is required to be passed in on BPX1MMS calls for function codes \_conn and \_free.

**Return\_code**

Returned parameter.

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_map\_service callable service stores the return code. The \_\_map\_service callable service stores a return code only if the return value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The \_\_map\_service callable service may return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EEXIST</b>	A request was made to perform a service on a block. However, either a map area is not active for the process or the map area is in the process of being shut down (JRMapNotActive).
<b>ENOMEM</b>	A request to create a new block or connect to an existing block was made with a zero block address, specifying that the __map service is to locate the address of a free map block. However, there are no unused blocks in the map area to satisfy the request (JRMapOutOfBlocks).
<b>EINVAL</b>	One of the following errors occurred: <ul style="list-style-type: none"> <li>• The FunctionCode parameter contains a value that is not a supported function, or the service call parameter list field MMG_SERVICETYPE contains an unsupported value (JRMapBadFunction).</li> <li>• A request was made to connect to a block, free the backing storage for a block, or change the access state (control operation) for a block. However, the token that was provided does not match that of any allocated block in the backing storage (JRMapTokenNotFound).</li> <li>• A MAP_NEWBLOCK or MAP_CONN request specified a map area block that is already in use (JRMapBlockInUse).</li> <li>• A request was made to connect to a block in the backing storage that is marked to be freed. The connection is not permitted (JRMapBlockFreePending).</li> <li>• A request was made to disconnect from a map block, but the block is not currently in use in the map area for this process (JRMapBlockNotInUse).</li> <li>• A reserved field contains nonzero data (JRReservedValueInvalid).</li> <li>• A block address was provided, but either it is not in the map area or it is not on a map block boundary (JRBadBlkAddr).</li> <li>• The array count was negative, zero, or greater than the maximum number of array elements permitted (1000) (JRMapArrayCountErr).</li> </ul>
<b>EFAULT</b>	An argument of this service contained an address that was not accessible to the caller (JRMapBadStorage).
<b>EMVSERR</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• There was an unexpected error (JRMapUnexpectedErr).</li> <li>• A request to create a new block, connect to an existing block, disconnect from an existing block, or change the read or write permissions for a block failed in RSM (JRIarvServ).</li> </ul>



**Reason\_code**

Returned parameter.

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_map\_service callable service stores the reason code. The \_\_map\_service callable service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. The \_\_map\_service callable service is designed to perform storage connects and disconnects very quickly. No data movement occurs.
2. Input to the \_\_map\_service callable service is an array of requests. Each request is processed in order until all requests have been successfully processed, or until an error occurs. When an error occurs, some requests may have been processed and some may not. An output flag on the array elements, \_\_mmg\_ReqFail, indicates the requests that have and have not been processed. The flag is off for array elements that have been processed successfully. The flag is on for the request that failed and all requests that had not yet been processed at the time of the failure.
3. The \_\_map\_service callable service allows an application to create a new data block and specify which map area block is to be used to view this data block. The map area block that is to contain the new data block must be free, that is, not currently connected to another data block. The kernel assigns a unique token to the new data block and returns this token to the application. The token is later used to identify the data block to subsequent calls to \_\_map\_service. The application may modify the new data block contained within its map block in any way it chooses.
4. Storage blocks are initially connected in write mode. When a block is in write mode, all worker processes that have the block connected have the block in write mode. If the block access is changed to read-only, all worker processes that have the block connected have the block in read-only mode.
5. Any areas within the map area that do not have a block connected are in the hidden state. Any reference to storage in the hidden state triggers a SIGSEVG signal.
6. If the initial process or a worker process forks, the child process inherits a map area that is initialized to the hidden state.
7. When an application has finished using a data block, it may do one of several things:
  - If it no longer needs the data block, it can disconnect it from the map block and request that the kernel free the data block. Once the data block has been freed and its use count has gone to zero, the data no longer exists in the kernel data space and is no longer available for processing.
  - If the data is still valuable, but is not currently needed, the application can request that the map area block be disconnected from the data block (without freeing it). This leaves the data block in a kernel data space for later use, while freeing the map area block for use in processing other data blocks. The map area block is hidden as part of the disconnect, and an OC4 abend occurs if the application attempts to reference any storage in the map area block.

Later, when a disconnected data block needs to be processed, the application can call \_\_map\_service with a connect request, specifying the token for the data block and the address of the map block it is to be attached to for processing. The \_\_map\_service callable service attaches the specified data block to the appropriate map block for use by the application. The block is read-only or read/write based on its state as of the last control operation.

- A data block may be freed without having first been connected by a call to \_\_map\_service with a free request, specifying the token of the data block.
- An application can control the access state (read or read/write) of a connected data block by calling \_\_map\_service with a control request and specifying the target state. Because special mechanisms

are used for the sharing of a data block between several processes, a state change is against the data block and affects all users of the data block (not just the current user's data block). State changes persist across disconnects. If a data block is made read-only and all users disconnect from the data block, the next user to connect to the data block obtains the block read-only.

See the description of the `__map_init` callable service, [“Usage notes” on page 415](#), for more information about using these two related services.

## Related services

- [“\\_\\_map\\_init \(BPX1MMI, BPX4MMI\) — Create a mapped megabyte area” on page 412](#)

## Characteristics and restrictions

Users of `__map_service` can create and manage a tremendous amount of data, causing the kernel to consume a large amount of system resources. To prevent abuse of such power, the `__map_init` service requires that the user be permitted to the BPX.MAP resource in the FACILITY class. (The `__map_service` callable service does not check for authority to BPX.MAP, because it does not perform any functions without first completing a `__map_init` request.).

## Examples

See [“BPX1MMS \(\\_\\_map\\_service\) example” on page 1262](#).

# mkdir (BPX1MKD, BPX4MKD) — Make a directory

---

## Function

The `mkdir` callable service creates a new, empty directory.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MKD)</b>	31-bit
<b>AMODE (BPX4MKD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MKD, (Pathname_length,
               Pathname,
               Mode,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MKD with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Fullword

The name of a fullword that contains the length of the full Pathname of the directory. The name can be up to 1023 bytes long. Each component of the name (between delimiters) can be up to 255 bytes long.

### Pathname

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Pathname\_length parameter.

The name of a field of length Pathname\_length that contains the full name of the directory.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

### Mode

Supplied parameter.

#### Type

Structure

#### Length

Fullword

The name of a fullword in which the mode field is specified. The mode field specifies the file type and the permissions you grant to yourself, to your group, and to any user.

The file type is identified using the BPXYFTYP mapping macro and permissions that are specified with the BPXYMODE mapping macro. See [“BPXYFTYP — File type definitions” on page 1078](#) and [“BPXYMODE — Map the mode constants” on page 1106](#).

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the mkdir service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mkdir service stores the return code. The mkdir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mkdir service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The process did not have search permission on some component of Pathname or did not have write permission on the parent directory of the directory to be created.
<b>EEXIST</b>	There is already a file or directory with the given Pathname. The following reason code can accompany the return code: JRMkDirExist.
<b>EFBIG</b>	A request to create a directory is prohibited because the file size limit for the process is set to 0.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMLINK</b>	The link count of the parent directory has already reached the maximum defined for the system. Refer to the LINK_MAX in “ <a href="#">pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name</a> ” on page 535 or to “ <a href="#">fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables</a> ” on page 233.
<b>ENAMETOOLONG</b>	Pathname contains more than 1023 characters; or a component of the name is longer than 255 characters.
<b>ENOENT</b>	Some component of Pathname does not exist or the Pathname parameter is blank.
<b>ENOSPC</b>	The file system does not have enough space to contain a new directory or the parent directory cannot be extended.
<b>ENOTDIR</b>	A component of Pathname is not a directory.
<b>EROFS</b>	The parent directory of the directory to be created is on a read-only file system. The following reason code can accompany the return code: JRMkDirROnly.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mkdir service stores the reason code. The mkdir service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The file permission bits that are specified through the Mode parameter are modified by the file creation mask of the calling process (see [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)). They are then used to set the file permission bits of the new directory.
2. The new directory's owner ID is set to the effective user ID (UID) of the calling process.
3. The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
  - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
  - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
4. The mkdir service sets the access, change, and modification times for the new directory. It also sets the change and modification times for the directory that contains the new directory.

## Related services

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)
- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)

## Characteristics and restrictions

None

## Examples

See [“BPX1MKD \(mkdir\) example” on page 1260](#).

# mkdirat (BPX1MDA, BPX4MDA) — Create a directory relative to a directory file descriptor

---

## Function

The mkdirat service operates in exactly the same way as the mkdir service, except for certain differences.

- If the path name given in Pathname is relative, then it is interpreted relative to the directory referred to by the file descriptor dirfd (rather than relative to the current working directory of the calling process, as is done by **mkdirat** for a relative path name).
- If Pathname is relative and dirfd is the special value AT\_FDCWD, then it is interpreted relative to the current working directory of the calling process (like **mkdirat**). If Pathname is absolute, then dirfd is ignored.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MDA)</b>	31-bit
<b>AMODE (BPX4MDA)</b>	64-bit
<b>ASC mode</b>	Primary mode

<b>Operation</b>	<b>Environment</b>
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MDA, (Pathname_length,
               Pathname,
               Mode,
               Dirfd,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MDA with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Character string

#### Character set:

No restriction

#### Length

Fullword

The name of a fullword that contains the length of the full Pathname of the directory. The name can be up to 1023 bytes long. Each component of the name (between delimiters) can be up to 255 bytes long.

### Pathname

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Pathname\_length parameter

The name of a field, of length Pathname\_length, that contains the full name of the directory.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts, based on the value of Dirfd. When Dirfd is AT\_FDCWD, the working directory is the current working directory. Otherwise, Dirfd contains the file descriptor for the starting directory.

### Mode

Supplied parameter.

#### Type

Structure

**Length**

Fullword

The name of a fullword in which the mode field is specified. The mode field specifies the file type and the permissions you grant to yourself, to your group, and to any user.

The file type is identified using the BPXYFTYP mapping macro and permissions that are specified with the BPXYMODE mapping macro. See [“BPXYFTYP — File type definitions”](#) on page 1078 and [“BPXYMODE — Map the mode constants”](#) on page 1106.

**Dirfd**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains a file descriptor for the starting directory to be used for the make node request. The special value AT\_FDCWD that was specified with a relative path name, indicates to use the current working directory to interpret the path. Otherwise, a relative pathname will be interpreted relative to the directory referred to by this file descriptor. If path name is absolute, this parameter is ignored.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mkdirat service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mkdirat service stores the return code. The mkdirat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mkdirat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process did not have search permission on some component of Pathname or did not have write permission on the parent directory of the directory to be created.
<b>EBADF</b>	Dirfd is not a valid file descriptor for AT_FDCWD.
<b>EEXIST</b>	There is already a file or directory with the given Pathname. The following reason code can accompany the return code: JRMkDirExist.
<b>EFBIG</b>	A request to create a directory is prohibited because the file size limit for the process is set to 0.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.

Return_code	Explanation
<b>EMLINK</b>	The link count of the parent directory has already reached the maximum defined for the system. Refer to the LINK_MAX in <a href="#">“pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name”</a> on page 535 or to <a href="#">“fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables”</a> on page 233.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the name is longer than 255 characters.
<b>ENOENT</b>	Some component of Pathname does not exist or the Pathname parameter is blank.
<b>ENOSPC</b>	The file system does not have enough space to contain a new directory or the parent directory cannot be extended.
<b>ENOTDIR</b>	Not a directory returned for the following: <ul style="list-style-type: none"> <li>• Some component of Pathname is not a directory (JrCompNotDir)</li> <li>• The dirfd parameter does not refer to a directory (JrAtNotDirFD)</li> </ul>
<b>EROFS</b>	The parent directory of the directory to be created is on a read-only file system. The following reason code can accompany the return code: JRMkDirROnly.

#### Reason\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the mkdirat service stores the reason code. The mkdirat service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The file permission bits that are specified through the Mode parameter are modified by the file creation mask of the calling process (see [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask”](#) on page 960). They are then used to set the file permission bits of the new directory.
2. The new directory's owner ID is set to the effective user ID (UID) of the calling process.
3. The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
  - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
  - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
4. The mkdirat service sets the access, change, and modification times for the new directory. It also sets the change and modification times for the directory that contains the new directory.

## Related services

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory”](#) on page 101
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory”](#) on page 420
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name”](#) on page 893
- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask”](#) on page 960



## Characteristics and restrictions

None.

### Examples

See “BPX1MDA (mkdirat) example” on page 1259 and “BPX4MDA (mkdirat) example” on page 1352.

# mknod (BPX1MKN, BPX4MKN) – Make a directory, a FIFO, a character special, or a regular file

## Function

The mknod callable service creates a new directory, a regular file, a character special file, or a FIFO special file (named pipe).

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MKN)</b>	31-bit
<b>AMODE (BPX4MKN)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MKN, (Pathname_length,
               Pathname,
               Mode,
               Device_identifier,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MKN with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Pathname of the special file to be created.

**Pathname**

Supplied parameter.

**Type**

Character string

**Character set**

No restriction

**Length**

Specified by the Pathname\_length parameter

The name of a field that contains the pathname of the file. The length of this field is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Mode**

Supplied parameter.

**Type**

Structure

**Length**

Fullword

The name of a fullword in which the mode field is specified. The mode field specifies the file type and the permissions you grant to yourself, to your group, and to any user. Specify the file type with the BPXYFTYP mapping macro, and specify permissions with the BPXYMODE mapping macro. See [“BPXYFTYP — File type definitions” on page 1078](#) and [“BPXYMODE — Map the mode constants ” on page 1106](#).

**Device\_identifier**

Supplied parameter.

**Type**

Structure

**Length**

Fullword

The name of a fullword that contains a device identifier, or 0. The high-order 16 bits of Device\_identifier is the device major number. The device major number corresponds to a device driver that supports a class of devices such as interactive terminals. The low-order 16 bits of Device\_identifier is the device minor number. The device minor number corresponds to a specific device within the class of devices that are referred to by the device major number. Specify Device\_identifier if you are creating a character special file.

If a FIFO, directory, or regular file is being created, Device\_identifier is ignored.

The following device major numbers are currently defined:

Device major numbers	Device class
1	Manager pseudoterminal
2	Subsidiary pseudoterminal
3	/dev/tty
4	/dev/null, /dev/zero, and /dev/random
5	/dev/fdn
6	Sockets

Device major numbers	Device class
7	OCSRTY
8	OCSADMIN
9	/dev/console

For device major numbers 1, 2, and 7, the device minor numbers refer to specific pseudoterminal pairs and the values range from 0 and one less than the maximum number of pseudoterminal pairs defined by the installation.

For device major numbers 3, 6, 8, and 9, the device minor number is ignored.

For device major number 4, device minor numbers represent files as follows:

Device minor numbers for device major number 4	File
0	/dev/null
1	/dev/zero
2	/dev/random and /dev/urandom

For device major number 5, the device minor number value represents the file descriptor to be referred to. For example, device minor 0 refers to file descriptor 0.

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the mknod service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the mknod service stores the return code. The mknod service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mknod service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process does not have permission to search some component of Pathname; or does not have write permission for the directory of the file to be created.
<b>EEXIST</b>	A file or directory named Pathname already exists. The following reason code can accompany the return code: JRSpFileExists.
<b>EFBIG</b>	A request to create a new file is prohibited because the file size limit for the process is set to 0.
<b>EINVAL</b>	The file type specified in the Mode parameter is not 1, 2, 3 or 4. The following reason code can accompany the return code: JRMknodInvalidType.

Return_code	Explanation
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters; or a component Pathname has a name longer than 255 characters.
<b>ENOENT</b>	A component of Pathname was not found; or no pathname was specified. The following reason code can accompany the return code: JREndingSlashMknod.
<b>ENOTDIR</b>	Not a directory returned for the following: <ul style="list-style-type: none"> <li>• Some component of Pathname is not a directory (JrCompNotDir)</li> <li>• The dirfd parameter does not refer to a directory (JrNotDirFd)</li> </ul>
<b>EPERM</b>	The operation is not permitted. The requested operation requires a superuser authority. The following reason code can accompany the return code: JrUserNotPrivileged.
<b>EROFS</b>	The directory of the file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFilesetMknodReq.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mknod service stores the reason code. The mknod service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The file permission bits of Mode are modified by the process's file creation mask (see “umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask” on page 960). They are then used to set the file permission bits of the file being created.
2. The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
  - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
  - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
3. The mknod service sets the access, change, and modification times for the new file. It also sets the change and modification times for the directory that contains the new file.

**Related services**

- “chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory” on page 101
- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 163
- “mknodat (BPX1MNA, BPX4MNA) — Make a directory, a FIFO, a character special, or a regular file relative to a directory file descriptor” on page 431
- “pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe” on page 551
- “stat (BPX1STA, BPX4STA) — Get status information about a file by path name” on page 893

- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)

## Characteristics and restrictions

When the mknodat service is requested to create a character special file, a directory or a regular file, it is a privileged operation and requires superuser authority.

## Examples

See [“BPX1MKN \(mknod\) example” on page 1261](#) and [“BPX4MKN \(mknod\) example” on page 1354](#).

# mknodat (BPX1MNA, BPX4MNA) — Make a directory, a FIFO, a character special, or a regular file relative to a directory file descriptor

---

## Function

The mknodat callable service operates in the same way as the mknod callable service, except for the following differences:

- If the path name given in Pathname is relative, then it is interpreted relative to the directory referred to by the file descriptor dirfd (rather than relative to the current working directory of the calling process, as is done by mknod() for a relative path name).
- If Pathname is relative and dirfd is the special value AT\_FDCWD, then it is interpreted relative to the current working directory of the calling process, like mknod(). If Pathname is absolute, then dirfd is ignored.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MNA)</b>	31-bit
<b>AMODE (BPX4MNA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MNA, (Pathname_length,
               Pathname,
               Mode,
               Device_identifier,
               Dirfd,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MNA with the same parameters. All parameter addresses in parameter structures are doublewords.

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the path name of the file whose mode you want to change.

### Pathname

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. The length of this field is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the directory pointed to by Dirfd. When Dirfd is AT\_FDCWD, the working directory is the current working directory. Otherwise, Dirfd contains the file descriptor for the starting directory.

### Mode

Supplied parameter.

#### Type

Structure

#### Length

Fullword

The name of a fullword in which the mode field is specified. The mode field specifies the file type and the permissions that you grant to yourself, to your group, and to any user. Specify the file type with the BPXYFTYP mapping macro, and specify permissions with the BPXYMODE-mapping macro. See [“BPXYFTYP – File type definitions” on page 1078](#) and [“BPXYMODE – Map the mode constants” on page 1106](#).

### Device\_identifier

Supplied parameter.

#### Type

Structure

#### Length

Fullword

The name of a fullword that contains a device identifier, or 0. The high-order 16 bits of Device\_identifier is the device major number. The device major number corresponds to a device driver that supports a class of devices such as interactive terminals. The low-order 16 bits of

Device\_identifier is the device minor number. The device minor number corresponds to a specific device within the class of devices that are referred to by the device major number. Specify Device\_identifier if you are creating a character special file.

If a FIFO, directory, or regular file is being created, Device\_identifier is ignored.

The following device major numbers are defined:

Device major numbers	Device class
1	Manager pseudoterminal
2	Subsidiary pseudoterminal
3	/dev/tty
4	/dev/null, /dev/zero, and /dev/random
5	/dev/fdn
6	Sockets
7	OCSRTY
8	OCSADMIN
9	/dev/console

For device major numbers 1, 2, and 7, the device minor numbers refer to specific pseudoterminal pairs and the values range from 0 and one less than the maximum number of pseudoterminal pairs defined by the installation.

For device major numbers 3, 6, 8, and 9, the device minor number is ignored.

For device major number 4, device minor numbers represent files as follows:

Device minor numbers for device major number 4	File
0	/dev/null
1	/dev/zero
2	/dev/random and /dev/urandom

For device major number 5, the device minor number value represents the file descriptor to be referred to. For example, device minor 0 refers to file descriptor 0.

Refer to [“mknod \(BPX1MKN, BPX4MKN\) — Make a directory, a FIFO, a character special, or a regular file” on page 427](#).

## Dirfd

Supplied parameter.

### Type

Integer

### Length

Fullword

The name of a fullword that contains a file descriptor for the starting directory to be used for the create directory request. The special value AT\_FDCWD that was specified with a relative path name, indicates to use the current working directory to interpret the path. Otherwise, a relative pathname will be interpreted relative to the directory referred to by this file descriptor. If path name is absolute, this parameter is ignored.

## Return\_value

Returned parameter.

### Type

Integer

### Length

Fullword

The name of a fullword in which the mknodat service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mknodat service stores the return code. It returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mknodat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process does not have permission to search some component of Pathname; or does not have write permission for the directory of the file to be created.
<b>EBADF</b>	Dirfd is not a valid file descriptor or AT_FDCWD.
<b>EFBIG</b>	A request to create a new file is prohibited because the file size limit for the process is set to 0.
<b>EINVAL</b>	The file type specified in the Mode parameter is not 1, 2, 3 or 4. The following reason code can accompany the return code: JRMknodInvalidType.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters; or a component Pathname has a name longer than 255 characters.
<b>ENOENT</b>	A component of Pathname was not found; or no pathname was specified. The following reason code can accompany the return code: JREndingSlashMknod.
<b>ENOTDIR</b>	Not a directory returned for the following: <ul style="list-style-type: none"> <li>• A component of Pathname is not a directory (JrCompNotDir).</li> <li>• The Dirfd parameter does not refer to a directory (JrAtNotDirFD).</li> </ul>
<b>EPERM</b>	The operation is not permitted. The requested operation requires a superuser authority. The following reason code can accompany the return code: JrUserNotPrivileged.
<b>EROFS</b>	The directory of the file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFilesetMknodReq.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mknodat service stores the reason code. The mknodat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



## Usage notes

1. The file permission bits of Mode are modified by the process's file creation mask (see [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)). They are then used to set the file permission bits of the file that is being created.
2. The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
  - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
  - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
3. The mknodat service sets the access, change, and modification times for the new file. It also sets the change and modification times for the directory that contains the new file.

## Related services

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“mknod \(BPX1MKN, BPX4MKN\) — Make a directory, a FIFO, a character special, or a regular file” on page 427](#)
- [“pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe” on page 551](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)
- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)

## Characteristics and restrictions

When the mknodat.service is requested to create a character special file, a directory or a regular file, it is a privileged operation and requires superuser authority.

## Examples

See [“BPX1MNA \(mknodat\) example” on page 1262](#) and [“BPX4MNA \(mknodat\) example” on page 1355](#).

# mmap (BPX1MMP, BPX4MMP) — Map pages of memory

---

## Function

The mmap callable service establishes a mapping between a process's address space and a zFS file.

## Requirements

Operation	Environment
Authorization	Supervisor state or problem state, PSW Key 2, or PSW Key 8
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1MMP)	31-bit
AMODE (BPX4MMP)	64-bit
ASC mode	Primary address space control (ASC) mode
Interrupt status	Enabled for interrupts
Locks	Unlocked

**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1MMP, (Map_address,
               Map_length,
               Protect_options,
               Map_type,
               File_descriptor,
               File_offset,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers need an additional parameter, `Returned_map_address`, as follows:

```
CALL BPX4MMP, (Map_address,
               Map_length,
               Protect_options,
               Map_type,
               File_descriptor,
               File_offset,
               Returned_map_address,
               Return_value,
               Return_code,
               Reason_code)
```

**Parameters****Map\_address**

Supplied parameter.

**Type**

Address

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains zero, or the address of an area within the address space at which the system is to attempt to map the requested file.

If the value of `Map_address` is zero, the system has complete freedom in selecting the location within the address space at which the requested file is mapped.

**BPX4MMP only:** The doubleword parameter allows a nonzero value in the high-order fullword. The Map Address and must not be in the range 2 G to 64 G.

**Map\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the size (in bytes) of the memory mapping that is to be created. The length that is specified must be less than or equal to the size of the file, and must not cause the address space REGION to be exceeded. Mapping operations are performed over whole pages, or whole segments when `MAP_MEGA` is specified. If the length is not a multiple of the page size or segment size, the entire trailing portion of the page or segment (up to the end of the file) is also mapped into the user storage. The trailing portion of the page or segment in which an end of file occurs contains binary zeros.

**BPX1MMP only:** If the high-order bit is on, the other 31 bits of the fullword parameter are interpreted as a pointer to a doubleword location. This doubleword is used as the 64-bit Map Length.

**BPX4MMP only:** The doubleword parameter allows a nonzero value in the high-order fullword.

### Protect\_options

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of the fullword that contains the value of the memory access protection flags. The `protect_options` parameter indicates whether read, write, execute, or some combination of accesses are permitted to the mapped data. Set it to either `PROT_NONE`, or a combination (by using, for example, an inclusive OR) of one or more of the other access protection flags. The constant values for these flags are defined in the `BPXYCONS` macro. (See “[BPXYCONS — Constants used by services](#)” on page 1062.) For `MAP_MEGA` mappings, the value that is specified for `protect_options` has a global effect on all current maps to the same file-offset range. For example, if `PROT_READ` is specified, all active maps have their protection for the same file-offset range changed to a protection of read.

Constant	Description
<b>PROT_READ</b>	Mapped data can be read. The file descriptor must have been previously opened with at least read access.
<b>PROT_WRITE</b>	Mapped data can be written and read. To select the <code>PROT_WRITE</code> option, if a <code>map_type</code> of <code>MAP_SHARED</code> is specified, the file descriptor must have been previously opened with read/write access. If <code>MAP_PRIVATE</code> is specified, the file descriptor only needs to have been opened with read access.
<b>PROT_EXEC</b>	Mapped data can be executed. This option is treated as if <code>PROT_READ</code> has been specified.
<b>PROT_NONE</b>	Mapped data cannot be accessed.

### Map\_type

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of the fullword that contains the mapping type. The constant values for `map_type` are defined in the `BPXYCONS` macro.

Constant	Description
<b>MAP_64</b>	Above the bar storage must be used for the map. If above the bar storage is not available for this address space, then an error is returned. <code>MAP_SHARED</code> is assumed and need not be specified.
<b>MAP_ANONYMOUS</b>	The mapping is not backed by any file and the memory contents are initialized to zero. When <code>MAP_ANONYMOUS</code> is set, the <code>fd</code> argument must be <code>-1</code> and the offset argument must be zero. <code>MAP_ANONYMOUS</code> can be used with any of the other <code>Map_type</code> options.
<b>MAP_FIXED</b>	The mapping must be placed at exactly the location that is specified by the <code>map_address</code> parameter.

Constant	Description
<b>MAP_MEGA</b>	All changes to the mapped data are shared. Modifications to the mapped data are visible to all other processes that map the same file-offset range. The protection attributes of file-offset ranges are common among all active maps. Changes to the protection option of a file-offset range are global, and immediately affect all active maps.
<b>MAP_PRIVATE</b>	All changes to the mapped data are private. Modifications to the mapped data are visible only to the calling process, and do not change the underlying file. To use this option, the hardware must provide the suppression-on-protection support.
<b>MAP_SHARED</b>	All changes to the mapped data are shared. Modifications to the mapped data are visible to all other processes that map the same file-offset range.

You must specify MAP\_SHARED, MAP\_PRIVATE, or MAP\_MEGA, but you cannot specify more than one. MAP\_FIXED is optional when any of the other map options is specified. To specify both MAP\_FIXED and MAP\_SHARED, for example, use a map\_type value equal to the inclusive OR of these two constants

### File\_descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the file descriptor of an open file that is to be mapped to process storage. The file descriptor is returned by [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#). You can only specify the file descriptor of a regular file.

For a MAP\_MEGA mapping, if this is the first map to the file that is represented by the specified file descriptor, the protect\_options that can be specified for this file by this map request (and by all future map or mprotect requests, by this or any other process mapping to the same file) are determined by whether the file was opened for read or for read and write.

- If the file was opened for read but not write, only PROT\_READ, PROT\_EXEC, or PROT\_NONE are allowed.
- If the file was opened for write, any of the protection options are accepted.

Once PROT\_WRITE is allowed for a file, all map requests must provide a file descriptor that was opened for write, or the map request is failed.

When MAP\_ANONYMOUS is specified, the File\_descriptor parameter must be -1.

### File\_offset

Supplied parameter.

#### Type

Integer

#### Length

Doubleword

The name of a doubleword that defines which part of the file is to be mapped. It contains the offset into the file at which the map\_length is to begin. The value of file\_offset must be a multiple of the page size when MAP\_PRIVATE or MAP\_SHARED is specified, and a multiple of the segment size when MAP\_MEGA is specified. The offset plus the map\_length must fall within the current size of the file.

When MAP\_ANONYMOUS is specified, the File\_offset parameter must be 0.

### Returned\_map\_address

Returned parameter (BPX4MMP only)

#### Type

Address

#### Length

Doubleword

The name of a doubleword in which the mmap service returns the 64-bit address where the mapping was placed, if the request is successful.

### Return\_value

Returned parameter

#### Type

Address

#### Length

Fullword

The name of a fullword in which the mmap service returns the 31-bit address at which the mapping was placed, if the request is successful; or -1, if it is not successful. In AMODE 64, if mmap is successful, 0 is returned in this field and the 64-bit address is returned in the Returned\_map\_address parameter.

Upon successful completion, the mmap service has established a mapping between the process's address space, at an address that is returned in the Return\_value parameter, for map\_length bytes, to the file that is represented by the file\_descriptor, at the specified file\_offset, for a length of map\_length bytes. The specified access protections and mapping type are set for the mapped range.

**BPX1MMP only:** If the request is successful and if MAP\_64 is specified or the high-order bit of the map\_address is set, then the 64-bit address where the mapping was placed is returned in the doubleword location that the map\_address points to and the Return\_Value is set to 0.

### Return\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the mmap service stores the return code. The mmap service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The mmap service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• The file descriptor is not open for read, regardless of the protection specified. (JRRFileNoRead)</li> <li>• The file descriptor is not open for write, and PROT_WRITE was specified for a MAP_SHARED type mapping. (JRWFileRDOOnly)</li> <li>• A MAP_MEGA request specified PROT_WRITE, but the first active map to a file was done with a file descriptor that was not open for write. (JRWFileMapRDOOnly)</li> </ul>
<b>EAGAIN</b>	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)

Return_code	Explanation
<b>EBADF</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• The file that is specified by the <code>file_descriptor</code> parameter does not represent a standard file. (JRNotStdFile)</li> <li>• The file that is specified by the <code>file_descriptor</code> parameter is not a valid open file descriptor. (JRs belong to <code>fstat()</code> or <code>w_ioclt()</code>)</li> </ul>
<b>EINVAL</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• <code>MAP_FIXED</code> was specified, and the requested range was not available. The range could be previously allocated, or it could be outside the address space region. (JRAddressNotAvailable).</li> <li>• <code>MAP_FIXED</code> was specified, the value of the <code>map_address</code> parameter is not a multiple of the page size. (JRNotPage).</li> <li>• The value of the <code>file_offset</code> parameter is not a multiple of the page size. (JRNotPage).</li> <li>• The value that is specified in the <code>map_type</code> parameter is incorrect. (JRMmapBadType).</li> <li>• The value that is specified in the <code>protect_options</code> parameter is incorrect. <code>PROT_NONE</code> cannot be specified in combination with any other options. (JROptNotSupp).</li> <li>• Subsequently, the file was extended and mapped beyond the original EOF point while an existing memory map containing the original EOF point was outstanding. (JRMmapOverEof).</li> <li>• The <code>file_offset</code> value must be zero or larger. (JRNegativeValueInvalid).</li> <li>• An attempt was made to map a file that is already mapped, but with a different specification of <code>MAP_MEGA</code>. At any point in time, a file can be mapped with or without the <code>MAP_MEGA</code> option, but not both with and without the <code>MAP_MEGA</code> option.</li> <li>• The file was already mapped by another process into a storage key that does not match the PSW key of the caller. (JrKeyMismatch).</li> <li>• An address greater than 31-bit <code>addr</code> but less than 64 G was passed in <code>map_address</code> (JrAddressNotAvailable).</li> <li>• When <code>MAP_ANONYMOUS</code> is specified, the <code>file_descriptor</code> parameter must be -1. (JrMmapAnonBadFd).</li> <li>• When <code>MAP_ANONYMOUS</code> is specified, the <code>file_offset</code> parameter must be 0. (JrMmapAnonBadOffset).</li> </ul>
<b>EMFILE</b>	<p>The number of mapped regions would exceed a system limit:</p> <ul style="list-style-type: none"> <li>• The system-wide limit on the amount of memory that is consumed by memory-mapped areas was exceeded. (JRMmapStgExceeded).</li> <li>• The per-process limit on the number of outstanding memory-mapped areas was exceeded. This limit is the same as the limit on the number of files a process can have open at any given time. (JRProcMaxMmap).</li> </ul>
<b>ENODEV</b>	<p>The file descriptor refers to a file for which <code>mmap</code> is not supported (for example, a terminal). (JRNotSupportedForFileType).</p>

Return_code	Explanation
<b>ENOMEM</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>MAP_FIXED was specified, and the requested range (map_address, map_address + map_length) exceeds that allowed for the address space of a process. (JRAccessNotAvailable).</li> <li>There is insufficient room in the address space to affect the mapping. (JRNoUserStorage).</li> <li>There is insufficient shared storage available in the system to satisfy this request. (JRShrStgStorage).</li> </ul>
<b>ENOSYS</b>	MAP_PRIVATE was specified, but the required suppression-on-protection hardware support was not available. (JRHardware).
<b>ENXIO</b>	The addresses in the range (file_offset, file_offset + map_length) are not valid for the specified file descriptor. (JRMmapFileAddress).

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mmap service stores the reason code. The mmap service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The mmap service supports only regular files. Any other type of file is not processed.
2. The mmap resources are maintained at a process level. This means that the termination of the thread that invoked the mmap service does not cause the associated mapping to be removed. The mmap resources are freed when the process ends.
3. The mmap service adds an extra reference to the file that is associated with the specified file descriptor that is not removed by a subsequent close on that file descriptor. This reference is removed when there are no more mappings to the file. The access level (read/write) that was established when the file was opened is enforced for the life of the memory-mapped area, independent of subsequent activity that occurs upon that file descriptor.
4. A file that is memory-mapped can be appended by another process while the memory map is active. No overlays will occur. However, the newly created area cannot be mapped across the original EOF point, unless either the EOF point falls on a 4 K boundary (1 M boundary, if MAP\_MEGA), or the original memory mapping is unmapped.
5. Specifying a target map\_address can have a negative impact on the address space. For example, specifying a map\_address at the top of the private area, below the 16 MB line, could prevent system code from successfully obtaining below-the-line storage.
6. All tasks and SRBs within the address space that issued the mmap request can access the memory that is allocated by the mmap service. However, only threads within the process that created the mmap area are permitted to invoke any subsequent memory map services against that mmap instance. The protection level that is established by this process is enforced for all accesses that are made to that range within the address space.
7. All memory-mapped areas, along with their mapping types and mprotect-established access levels, are propagated to the child process during fork processing. The user is responsible for serialization across multiple threads.

8. If MAP\_PRIVATE is specified, the initial write reference to the memory-mapped region creates a private copy of the memory-mapped page, and redirects the mapping to the copy. The copy is not created until the first write. Until the first write, updates that are made to that region by other processes that are mapped by MAP\_SHARED with the same file-offset range are visible.
9. Applications that use the MAP\_PRIVATE support might need to be aware of page boundaries when updates are performed. An update to a single byte causes an entire page to no longer receive updates that are made by other processes that are mapped with the same file-offset range.
10. To serialize access to a file-offset range that is being accessed by multiple processes, you can use lockf, fcntl, or semaphores. Serialization should be obtained when the incore copy of the data is being updated, or when the file is being updated using msync.
11. If a sparse file is memory-mapped, accessing a page that has never been written to in the file causes a page of binary zeros to be generated.
12. The mmap service allows access to zFS files through address space manipulation, instead of through the read/write services. After the file is mapped, the process can access it by using the data at the address to which the file was mapped.

The following code sample illustrates how an existing program might be changed to use the mmap service:

```
fd = open(...)
lseek(fd, file_offset)
read(fd, buffer, length)

/* ...(use data in buffer) ... */

becomes

fd = open(...)
address = mmap (0, length, PROT_READ, MAP_PRIVATE, fd, file_offset)

/* ...(use data at address) ... */
```

13. Constants that are used for this callable service are defined in the BPXYCONS macro. See [“BPXYCONS — Constants used by services”](#) on page 1062.
14. If MAP\_64 is specified, then the map\_address parameter must either be 0 or a value greater than 64G or EINVAL will be returned.
15. If both MAP\_64 and MAP\_FIXED are specified, then the first fullword of the map\_address parameter must be nonzero or EINVAL will be returned.
16. For non-MAP\_FIXED requests, if the map\_address specifies an address above 64 G, the system attempts to create a mapping at the address that is specified by map\_address and the address must be on a segment boundary. If it is unsuccessful, it proceeds as if a map\_address value of zero were specified but will attempt to create a mapping above 64 G.

## Related services

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216
- [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file”](#) on page 249
- [“mprotect \(BPX1MPR, BPX4MPR\) — Set protection of memory mapping ”](#) on page 452
- [“msync \(BPX1MSY, BPX4MSY\) — Synchronize memory with physical storage ”](#) on page 469
- [“munmap \(BPX1MUN, BPX4MUN\)— Unmap previously mapped addresses ”](#) on page 473
- [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ”](#) on page 574
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“semget \(BPX1SGT, BPX4SGT\) — Create or find a set of semaphores”](#) on page 718
- [“setrlimit \(BPX1SRL, BPX4SRL\) — Set resource limits ”](#) on page 789
- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ”](#) on page 913



## Characteristics and restrictions

1. For MAP\_FIXED requests, the value of map\_address must be a multiple of the page size when MAP\_PRIVATE or MAP\_SHARED is specified, and a multiple of the segment size when MAP\_MEGA or MAP\_64 is specified
2. A MAP\_FIXED request fails with an EINVAL if any portion of the requested range is already in use for any reason (including a previous mapping).
3. If MAP\_MEGA is specified, the value that is specified in map\_address must be equal to zero or equal to or greater than 16 megabytes, or the request is failed with EINVAL
4. If MAP\_64 is specified or a map\_length has a value greater than 2 G, the value that is specified in map\_address must be equal to zero or equal to or greater than 64 G, or the request is failed with EINVAL.
5. The MAP\_PRIVATE support requires the suppression-on-protection hardware feature.
6. The same file-offset range can be mapped multiple times within a given address space (to different virtual addresses), each with unique protection levels. A memory-mapped file-offset range can partially or fully overlap other existing mapped file-offset ranges. This support also holds true across multiple processes.
7. The mmap service can never be used to extend or truncate the size of a file. If a page is updated beyond the EOF mark of the original memory-mapped file, the portion beyond the EOF mark is not written to the file.
8. A file that is memory-mapped can be appended by another process while the memory map is active. No overlays will occur. However, the newly created area cannot be mapped across the original EOF point unless either the EOF point falls on a page boundary (segment boundary, if MAP\_MEGA) or the original memory mapping is unmapped.
9. When a given file-offset is memory-mapped, unpredictable results will occur if the file is truncated to a point that resides within the memory mapped range. These results might include the abnormal termination of the task that is accessing the memory-mapped area.
10. If other processes modify the contents of the file that is using the write service while mapped ranges are active for that file-offset, results will be unpredictable, unless specific serialization actions are taken by the user. See [“msync \(BPX1MSY, BPX4MSY\) – Synchronize memory with physical storage ” on page 469](#) for details.
11. There is a limit on the number of active memory maps that a process can have outstanding at any given time. The system administrator defines this limit by specifying the maximum number of files a process can have open. Even though a single value is set that limits both files and mmaps, the two limits are enforced independently of one another.
12. Memory maps with the MAP\_MEGA option use storage in units of megabytes. Extensive use of MAP\_MEGA on very small files, or on small ranges of larger files, can be wasteful. MAP\_MEGA is best used on large files.
13. Memory maps of very large files by several processes can realize substantial savings of system common area usage when you use the MAP\_MEGA option.
14. Specify 64-bit support by using the MAP\_64 option of Map\_type, providing a Map\_address above 64 G or a Map\_length greater than 2 G.
15. If the first mapping of a file uses the 64-bit support, then all subsequent mappings of the file must also use 64-bit support until there are no more mappings onto that file. If an attempt is made to map the file without using 64-bit support, an EINVAL error is returned.
16. If the first mapping of a file uses the 31-bit support, then all subsequent mappings of the file must also use 31-bit support until there are no more mappings onto that file. If an attempt is made to map the file without using 31-bit support, an EINVAL error is returned.

## Examples

See [“BPX1MMP \(mmap\) example” on page 1262](#).

## mount (BPX1MNT) – Make a file system available

---

### Function

The mount callable service mounts a file system, making the files in it available for use.

**Note:** There is no 64-bit version of the mount callable service. To get equivalent function, use “[\\_\\_mount](#) (BPX2MNT, BPX4MNT) – Make a file system available ” on page 447 in 64-bit mode.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE</b>	31-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1MNT, (MountPoint_length,
               MountPoint_name,
               File_system_name,
               File_system_type,
               Mount_mode,
               Parm_length,
               Parm,
               Return_value,
               Return_code,
               Reason_code)
```

### Parameters

#### MountPoint\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of MountPoint\_name.

#### MountPoint\_name

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

**Length**

Specified by the MountPoint\_length parameter

The name of a field that contains the name of the mount point. The length of this field is specified in MountPoint\_length.

**File\_system\_name**

Supplied parameter.

**Type**

Character string

**Character set**

Printable characters

**Length**

44 bytes

The name of a 44-character field that identifies the file system to be mounted. The name must be left-aligned and padded with blanks.

**File\_System\_type**

Supplied parameter.

**Type**

Character string

**Character set**

Printable characters

**Length**

8 bytes

The name of a field that contains the 8-character file system type. This corresponds to the type of file system that was defined by a FILESYSTYPE parameter of the BPXPRMxx parmlib member.

**Mount\_mode**

Supplied parameter.

**Type**

Structure

**Length**

Fullword

The name of a fullword that contains binary flags. The flags can indicate any of the following situations.

- The mount mode (read or read/write).
- Whether the mount request must complete synchronously.
- Whether SETUID is not allowed.
- NOSECURITY.

This parameter is mapped by the macro BPXYMTM; see [“BPXYMTM — Map the modes for mount and unmount”](#) on page 1110 for details.

**Parm\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of the file system-specific parameters (Parm). The maximum length is 500 bytes.

**Parm**

Supplied parameter.

The name of a field, of length `Parm_length`, that contains the file-system-specific parameters. These have a maximum of 500 bytes.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mount service returns 0 or 1 if the request is successful, or -1 if it is not successful. A `Return_value` of 1 indicates that the mount will complete asynchronously.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mount service stores the return code. The mount service always returns `Return_code` if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mount service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBUSY</b>	The file system to mount is quiesced; or no more locks are available. The following reason codes can accompany the return code: <code>JROutOfLocks</code> , <code>JRQuiesced</code> .
<b>EINVAL</b>	<p>There was a parameter error. Verify the <code>Mount_mode</code> and <code>File_system_type</code>. Other reasons for this error include:</p> <ul style="list-style-type: none"> <li>• The mount point is a root of a file system.</li> <li>• The file system is already mounted.</li> <li>• <code>parm_length</code> is too long.</li> <li>• A mounted file system has a real or alias name that conflicts with this mount request. One of these situations occurred: <ul style="list-style-type: none"> <li>– A file system was previously mounted using an alias data set name, and the corresponding real data set name conflicts with the file system name specified by this mount request.</li> <li>– The file system name specified on this mount request is an alias data set name that has a real data set name which conflicts with the name of a previously mounted file system. Resolve the duplicate file system names.</li> </ul> </li> </ul> <p>Resolve the duplicate file system names and reissue the mount request.</p> <p>The following reason codes can accompany the return code: <code>JROutOfLocks</code>, <code>JRQuiesced</code>, <code>JRIsMountedRealName</code>.</p>
<b>EIO</b>	An I/O error occurred.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the <code>Pathname</code> argument. This error is issued if more than 24 symbolic links are detected in the resolution of <code>Pathname</code> .

Return_code	Explanation
<b>ENOENT</b>	The mount point does not exist. The following reason code can accompany the return code: JRMountPt.
<b>ENOMEM</b>	There is not enough storage space available to mount this file system.
<b>ENOTDIR</b>	The mount point is not a directory. The following reason code can accompany the return code: JRMountPt.
<b>EPERM</b>	Insufficient authority to do the mount.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mount service stores the reason code. The mount service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. The caller must be APF-authorized or must be running for a user with mount authority.
2. The mount service effectively creates a virtual file system. After a file system is mounted, references to the path name that is mounted refer to the root directory on the mounted file system.
3. A file system can be mounted at only one point.
4. The physical file system might complete the mount operation asynchronously, which is indicated by a Return\_value of 1. The w\_getmntent callable service can then be used to determine whether the file system was mounted.

**Related services**

- [“umount \(BPX1UMT, BPX4UMT\) — Remove a virtual file system” on page 961](#)
- [“w\\_getmntent \(BPX1GMN, BPX4GMN\) — Get information on mounted file systems” on page 995](#)

**Characteristics and restrictions**

1. In order to mount a file system, the caller must be an authorized program, or must be running for a user with appropriate privileges (see [“Authorization” on page 8](#)).
2. Typically, an EBUSY error condition is returned when a file system is quiesced. In a sysplex the mount syscall is suspended until the file system becomes unquiesced.

**Examples**

See [“BPX1MNT \(mount\) example” on page 1263](#).

## **\_\_mount (BPX2MNT, BPX4MNT) — Make a file system available**

**Function**

The \_\_mount callable service mounts a file system, making the files in it available for use.

## Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX2MNT)	31-bit
AMODE (BPX4MNT)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX2MNT, (Mnte_length,  
               Mnte,  
               Return_value,  
               Return_code,  
               Reason_code)
```

AMODE 64 callers use BPX4MNT with the same parameters.

## Parameters

### Mnte\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of Mnte and its associated data structures, such as parameter string length.

### Mnte

Supplied parameter

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Mnte\_length parameter

The MNTNTE data structure. This is composed of a header field, the body field, and an additional area for the parameter string if one is being used. This structure is mapped by BPXYMNTNTE (see [“BPXYMNTNTE — Map response and element structure of w\\_getmntent” on page 1104](#)). See the usage notes for the fields in this data structure that must be set for the different \_\_mount requests.

### Return\_value

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the \_\_mount service returns 0 or 1 if the request is successful, or -1 if it is not successful. A Return\_value of 1 indicates that the mount will complete asynchronously.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the \_\_mount service stores the return code. The \_\_mount service always returns Return\_code if Return\_value is -1. For a list of return code values, see *Return codes (errno)* in *z/OS UNIX System Services Messages and Codes*. The \_\_mount service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCESS</b>	The user who is doing this nonprivileged mount does not have access to either the mount point or root. The following reason code can accompany the return code: JRNoMntPtAccess, JRNoRootAccess.
<b>EBUSY</b>	The file system to be mounted is quiesced; or no more locks are available. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced.
<b>EINVAL</b>	<p>There was a parameter error. Verify the Mount_mode and File_system_type (specified in the MNT data structure). Other reasons for this error include:</p> <ul style="list-style-type: none"> <li>• The mount point is a root of a file system.</li> <li>• The file system is already mounted.</li> <li>• The parm_length is too long.</li> <li>• A mounted file system has a real or alias name that conflicts with this mount request. One of these situations occurred: <ul style="list-style-type: none"> <li>– A file system was previously mounted using an alias data set name, and the corresponding real data set name conflicts with the file system name specified by this mount request.</li> <li>– The file system name specified on this mount request is an alias data set name that has a real data set name that conflicts with the name of a previously mounted file system. Resolve the duplicate file system names.</li> </ul> </li> </ul> <p>Resolve the duplicate file system names and reissue the mount request.</p> <p>The following reason codes can accompany the return code if it is a nonprivileged user mount: JRNotSupportedForFileType, JRFileSystemMigrated, JRSecurityConflict, JRNoSetUID, JRSysNameNotAllowed, JRPFSNotSupported.</p> <p>The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced, JRIsMountedRealName.</p>
<b>EIO</b>	An I/O error occurred.

<b>Return_code</b>	<b>Explanation</b>
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENOENT</b>	The mount point does not exist. The following reason code can accompany the return code: JRMOUNTPT.
<b>ENOMEM</b>	There is not enough storage space available to mount this file system.
<b>ENOTDIR</b>	The mount point is not a directory. The following reason code can accompany this return code: JRMOUNTPT.
<b>ENOTEMPTY</b>	The mount point directory is not empty for the nonprivileged user mount.
<b>EPERM</b>	Insufficient authority to do the mount.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the \_\_mount service stores the reason code. The \_\_mount service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The caller must be APF-authorized or must be running for a user with mount authority.
2. The \_\_mount service effectively creates a virtual file system. After a file system is mounted, references to the path name that is mounted refer to the root directory on the mounted file system.
3. A file system can be mounted at only one point.
4. The File\_system\_name value must be uppercase, and must be the name of the data set.
5. The physical file system may complete the \_\_mount operation asynchronously, which is indicated by a Return\_value of 1. The w\_getmntent callable service can then be used to determine if the file system has been mounted.

**Usage notes for eye catchers**

The MNTE eye-catcher must be set to MNT2. The length of the body following the header must also be set into MNTEHLEN, the body length field in the header.

- When setting character string values like SYSNAME or FROMSYSNAME, in general, these fields are set with the strings left-aligned and blank padded. Should data in these fields be unrequired or absent, the values should be set to nulls. Consult the data structure definition for specifics.
- When requesting a change to an already mounted file systems, the MNTENTCHANGE bit must be set on. Additionally, when requesting that the AUTOMOVE setting be changed, the MNTENTNEWAUTO bit must be set on.
- When requesting a mount of a file system, as opposed to a change, none of the MNTENTRFLAGS are expected to be set on.
- When requesting that a collection of file systems be moved from one system to another, the following fields must be set:
  - **FROMSYS** (to indicate where the file systems are to be moved from).
  - **SYSNAME** (to indicate where the file systems are to be relocated).



- **Rflags** (to indicate that this is a change mount request).

The other fields are ignored.

- When requesting a single file system move, the mount point or the file system name must be specified. Do not specify both. Additionally, the name of the system that the file system should be specified in SYSNAME. If you plan to change an AUTOMOVE setting, set the new value in the bit of the FSmode word. The Rflags setting will specify that MNTENTCHANGE=ON, which indicates that the change is a **chmount** request. Set MNTENTNEWAUTO only if the request intends to change the AUTOMOVE setting to what is reflected in the MNTENTFSNOAUTOMOVE value.

For more information about SYSNAME and AUTOMOVE, see [Customizing BPXPRMxx for a shared file system in z/OS UNIX System Services Planning](#). The chmount command is explained in [chmount - Change the mount attributes of a file system in z/OS UNIX System Services Command Reference](#).

- When requesting a mount on a system other than the one the mount command is executed on, the MNTENTSYSNAME field will denote the system that will "own" the file system. Fields that must be set to request that a mount are Filemode settings (such as read or write), FILESYSNAME, FILESYSTYPE, and *pathname*. Other fields that can be optionally set are parameter string and *systemname*.

The following types of mounts are supported when using an MNT eye catcher of MNT3. The length of the body following the header must also be set into MNTHEBLEN. For more information about the mount types, see [mount - Logically mount a file system in z/OS UNIX System Services Command Reference](#).

#### Bind mount

The MntEntFSModeBind flag must be set and the source path name and path length must be specified in MntEntSrcPathOffset and MntEntSrcPathLength.

#### Remount

The MntEntFSModeRemount flag must be set and MntEntMountPoint must be set to an existing mount point. The value of MntEntFSModeRdOnly determines the mode of the mounted file system. This type of mount is similar in behavior to **chmount**.

#### Bind with remount

The MntEntFSModeBind and MntEntFSModeRemount flags must be set and MntEntMountPoint must be set to an existing mount point. The value of MntEntFSModeRdOnly determines the mode of the mounted file system.

#### Move mount

The MntEntFSModeMove flag must be set and the source path name and path length must be specified in MntEntSrcPathOffset and MntEntSrcPathLength.

#### Update mount propagation

The MntEntFSModePrivate or MntEntFSModeUnbindable flags must be set and MntEntMountPoint must be set to an existing mount point.

## Related services

- [“umount \(BPX1UMT, BPX4UMT\) — Remove a virtual file system” on page 961](#)
- [“w\\_getmntent \(BPX1GMN, BPX4GMN\) — Get information on mounted file systems” on page 995](#)

## Characteristics and restrictions

1. In order to mount a file system, the caller must be an authorized program, or must be running for a user with mount authority. For more information, see [Mounting file systems in z/OS UNIX System Services Planning](#).
2. Typically, an EBUSY signal is returned when a file system is quiesced. In a sysplex, the \_\_mount() syscall suspends until the file system becomes unquiesced.

## Examples

See [“BPX2MNT \(\\_\\_mount\) example” on page 1263](#).

## mprotect (BPX1MPR, BPX4MPR) – Set protection of memory mapping

---

### Function

The mprotect callable service changes the access protection of a memory mapping for the caller's address space.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, PSW Key 2 or PSW Key 8
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MPR)</b>	31-bit
<b>AMODE (BPX4MPR)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1MPR, (Map_address,
               Map_length,
               Protect_options,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MPR with the same parameters. The Map\_address and Map\_length parameters are doublewords.

### Parameters

#### Map\_address

Supplied parameter.

#### Type

Address

#### Length

Fullword (doubleword).

The name of a fullword (doubleword) that contains the starting address in the address space at which the access protection of the mapping is to be changed. The value of Map\_address must be a multiple of the page size.

**BPX1MPR only:** If the high-order bit is on, the other 31 bits of the fullword parameter is interpreted as a pointer to a doubleword location. This doubleword is used as the 64-bit map address and must not be in the range 2 G to 64 G.

**BPX4MMP only:** The doubleword parameter allows a nonzero value in the high-order fullword.

**Map\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword (doubleword)

The name of the fullword (doubleword) that contains the size (in bytes) of the mapping that is to have its access protection modified. The length can be the size of the whole mapping, or a part of it. If the specified length is not in multiples of the page size, it is rounded up to a page boundary.

**BPX1MPR only:** If the high-order bit is on, the other 31 bits of the fullword parameter is interpreted as a pointer to a doubleword location. This doubleword is used as the 64-bit Map length.

**BPX4MMP only:** The doubleword parameter allows a nonzero value in the high-order fullword. The Map Address and must not be in the range 2 G to 64 G.

**Protect\_options**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of the fullword that contains the new value of the access protection flags for the specified mapping. You can change the access protection flags to either PROT\_NONE or a combination (for example, by using an inclusive OR) of one or more of the other flags (such as PROT\_READ, PROT\_WRITE, or PROT\_EXEC). These flags are defined in the BPXYCONS macro. (See [“BPXYCONS — Constants used by services”](#) on page 1062.)

Constant	Description
<b>PROT_READ</b>	Mapped data can be read.
<b>PROT_WRITE</b>	Mapped data can be written and read.
<b>PROT_EXEC</b>	Mapped data can be executed. PROT_EXEC is treated in the same way as PROT_READ.
<b>PROT_NONE</b>	Mapped data cannot be accessed.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mprotect service returns the value of 0 if the request is successful, or -1 if it is not successful.

Upon successful completion, the mprotect service has changed the access protections on the mapping that is specified by the range (map\_address, map\_address + map\_length) to those specified by the protect\_options parameter.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the mprotect service stores the return code. The mprotect service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The mprotect service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The access protection value is incorrect; it violates the access permission of the process to the underlying file. The following condition occurred: <ul style="list-style-type: none"> <li>• The original file is not open for write, and PROT_WRITE is specified for a MAP_SHARED type mapping.</li> </ul>
<b>EAGAIN</b>	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey).
<b>EINVAL</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• The value of map_address is not a multiple of the page size. (JRNotPage).</li> <li>• The input address or length is negative. (JRNegativeValueInvalid).</li> <li>• The specified protection options are not valid. (JROptNotSupp).</li> <li>• The caller's PSW key does not match the key of the memory mapped storage segment that is being operated against. (JrKeyMismatch).</li> <li>• In 64-bit mode, the value of map_address specified was greater than X'7FFFFFFFFF' (JrAddressNotAvailable).</li> <li>• In 64-bit mode, the value of map+length was greater than X'7FFFFFFFFF' (Jr_InvParmLength).</li> </ul>
<b>ENOMEM</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• Addresses in the range (map_address, map_address + map_length) are not valid for the address space. (JRAddressNotAvailable).</li> <li>• One or more specified pages are not mapped. (JRNotMapped).</li> </ul>

#### **Reason\_code**

Returned parameter.

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword in which the mprotect service stores the reason code. The mprotect service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### **Usage notes**

1. Access protection only acts on full pages. If the map\_length parameter contains a value that is not a multiple of the page size, the length is rounded up to a full page.
2. The protection level that is established by the mprotect service is address-space wide in scope, not just process specific. The scope is system-wide when the protection is changed for a MAP\_MEGA map. All active maps to the same file-offset range are affected by the request.
3. Constants that are used for this callable service are defined in the BPXYCONS macro. See [“BPXYCONS – Constants used by services”](#) on page 1062.

## Related services

- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ” on page 913](#)
- [“mmap \(BPX1MMP, BPX4MMP\) — Map pages of memory ” on page 435](#)
- [“msync \(BPX1MSY, BPX4MSY\) — Synchronize memory with physical storage ” on page 469](#)
- [“munmap \(BPX1MUN, BPX4MUN\)— Unmap previously mapped addresses ” on page 473](#)

## Characteristics and restrictions

The range specified (map\_address, map\_address + map\_length) must not contain any areas that are not memory-mapped. However, it may contain areas that have been unmapped, in which case no action is taken against the unmapped areas.

## Examples

See [“BPX1MPR \(mprotect\) example” on page 1265](#).

# msgctl (BPX1QCT, BPX4QCT) – Perform message queue control operations

---

## Function

The msgctl service provides a variety of message control operations as specified by the Command parameter. These functions include reading and changing message variables within the MSQID\_DS data structure, and removing a message queue from the system.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1QCT):</b>	31-bit
<b>AMODE (BPX4QCT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax is as follows:

```
CALL BPX1QCT, (Message_Queue_ID,
               Command,
               Buffer,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4QCT with the same parameters. The Buffer parameter is a doubleword.

## Parameters

### Message\_Queue\_ID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

Specifies the message queue identifier.

### Command

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that indicates the message command to be executed. For the structure that contains these constants, see [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098. The values for Command are:

**Ipc\_STAT**

Obtain status information about the message queue that is identified by the Message\_Queue\_ID parameter, if the current process has read permission. This information is stored in the area that is pointed to by argument Buffer and mapped by area MSQID\_DS data structure. For the data structure, see [“BPXYMSG — Map interprocess communication message queues”](#) on page 1107, MSQID\_DS DSECT.

**Ipc\_STAT\_EX**

When the AMODE of the caller is 31, it provides the same output as IPC\_STAT when AMODE is 64 (that is, extended POSIX times are provided). When the AMODE of the caller is 64, IPC\_STAT\_EX and IPC\_STAT are equivalent.

**Ipc\_SET**

Set the value of the IPC\_UID, IPC\_GID, IPC\_MODE and MSG\_QBYTES for associated Message\_queue\_ID. The values that are to be set are taken from the MSQID\_DS data structure that is pointed to by argument Buffer. Any value for IPC\_UID and IPC\_GID can be specified. Only mode bits that are defined by msgctl under Message\_Flag argument can be specified in the IPC\_MODE field. This Command can only be executed by a task that has an effective user ID equal either to that of a task with appropriate privileges (see [“Authorization”](#) on page 8), or to the value of IPC\_CUID or IPC\_UID in the MSQID\_DS data structure that is associated with Message\_Queue\_ID. This information is taken from the buffer that is pointed to by the Buffer parameter. For the data structure, see [“BPXYMSG — Map interprocess communication message queues”](#) on page 1107, MSQID\_DS DSECT.

**Ipc\_SET\_EX**

When the AMODE of the caller is 31, it assumes the same input as IPC\_SET when AMODE is 31 and provides the same output as IPC\_SET when AMODE is 64 (that is, extended POSIX times are provided). When the AMODE of the caller is 64, IPC\_SET\_EX and IPC\_SET are equivalent.

**Ipc\_RMID**

Remove the message identifier that is specified by Message\_Queue\_ID from the system, and destroy the message queue and MSQID\_DS data structure that are associated with it. This Command can only be executed by a process that has an effective user ID equal either to that of a process with appropriate privileges (see [“Authorization”](#) on page 8), or to the value of IPC\_CUID or IPC\_UID in the MSQID\_DS data structure that is associated with Message\_Queue\_ID.

**Buffer**

Parameter supplied and returned

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of the fullword (doubleword) that contains the address of the buffer into which or from which the message queue information will be copied. This buffer is mapped by MSQID\_DS. (See [“BPXYMSG — Map interprocess communication message queues”](#) on page 1107.)

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgctl service returns -1 or 0.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgctl service stores the return code. The msgctl service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The msgctl service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The command specified was Ipc_STAT, and the calling process does not have read permission. The following reason code can accompany the return code: JRIPcDenied.
<b>EINVAL</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• Message_Queue_ID is not a valid Message queue identifier.</li> <li>• The Command parameter is not a valid command.</li> <li>• The mode bits were not valid (SET). The following reason codes can accompany the return code: JRIPcBadFlags, JRMsqQBytes, or JRIPcBadID.</li> </ul>
<b>EPERM</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• The command specified was Ipc_RMID or Ipc_SET. The effective user ID of the caller is not that of a process with appropriate privileges (see <a href="#">“Authorization”</a> on page 8), and is not the value of IPC_CUID or IPC_UID in the MSQID_DS data structure that is associated with Message_Queue_ID.</li> <li>• The command specified was Ipc_SET, and an attempt is being made to increase MSG_QBYTES. The effective user ID of the caller does not have superuser privileges. The following reason codes can accompany the return code: JRIPcDenied or JRMsqQBytes.</li> </ul>
<b>EFAULT</b>	The Buffer parameter specified an address that caused the syscall to program check. The following reason code can accompany the return code: JRBadAddress.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgctl service stores the reason code. The msgctl service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Changing the access permissions only affects message queue syscall requests that occur after the msgctl service has returned. The msgsnd and msgrcv services, which are waiting while the permission bits are changed by msgctl, are not affected.
2. Ipc\_SET can change permissions, and can affect the ability of a thread to use the next message queue syscall.
3. Quiescing a message queue stops additional messages from being added, while allowing existing messages to be received. You can quiesce a message queue by clearing (Ipc\_SET) write permission bits.
4. You can also quiesce a message queue by reducing MSG\_QBYTES (Ipc\_SET) to zero. (It would take a superuser to reraise the limit.) Requesters are told EAGAIN or wait.
5. When a message queue ID is removed (Ipc\_RMID) from the system, all waiting threads regain control with RV=-1, RC=EIDRM, and RC=JRIPCRemoved.
6. If you do not want to change all the fields, first initialize (Ipc\_STAT) the buffer, change the desired fields, and then make the change (Ipc\_SET).
7. For Command Ipc\_RMID, the remove is complete by the time control returns to the caller.

**Related services**

- [“msgget \(BPX1QGT, BPX4QGT\) — Create or find a message queue” on page 458](#)
- [“msgrcv \(BPX1QRC, BPX4QRC\) — Receive from a message queue ” on page 462](#)
- [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue ” on page 465](#)

**Characteristics and restrictions**

The invoker is restricted by ownership, read, and read/write permissions defined by msgget and msgctl Ipc\_SET.

**Examples**

See [“BPX1QCT \(msgctl\) example” on page 1276](#).

## **msgget (BPX1QGT, BPX4QGT) — Create or find a message queue**

---

**Function**

The msgget function returns a message queue ID that it created or that the user is allowed to access.



## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1QGT):</b>	31-bit
<b>AMODE (BPX4QGT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1QGT, (Key,
               Message_Flag,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4QGT with the same parameters.

## Parameters

### Key

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Identification for this message queue. This can be a user-defined value that serves as a lookup value to determine if this message queue already exists, or the reserved value `IpC_PRIVATE`.

### Message\_Flag

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Valid values for this field include any combination of the following (additional bits cause an `EINVAL`):

#### IpC\_CREAT

Creates a message queue if the key that is specified does not already have an associated ID. `IpC_CREATE` is ignored when `IpC_PRIVATE` is specified.

#### IpC\_EXCL

Causes the `msgget` function to fail if the key that is specified has an associated ID. `IpC_EXCL` is ignored when `IpC_CREAT` is not specified, or when `IpC_PRIVATE` is specified.

**IpC\_RcvTypePID**

Creates a message queue that can only be read from (by the msgrcv service) when Message\_Type is the process ID of the invoker. This restriction does not apply if the caller of the msgrcv service has the same effective UID as the creator of the message queue.

**IpC\_SndTypePID**

Creates a message queue that can only be written to (by the msgsnd service) when Message\_Type is the process ID of the invoker. This restriction does not apply if the caller of the msgsnd service has the same effective UID as the creator of the message queue.

**IpC\_PL01**

Use PLO for serialization.

**IpC\_PL02**

Use PLO if practical.

**S\_IRUSR**

Permits the process that owns the message queue to read it.

**S\_IWUSR**

Permits the process that owns the message queue to alter it.

**S\_IRGRP**

Permits the group that is associated with the message queue to read it.

**S\_IWGRP**

Permits the group that is associated with the message queue to alter it.

**S\_IROTH**

Permits others to read the message queue.

**S\_IWOTH**

Permits others to alter the message queue.

The values that begin with an "IpC\_" prefix are defined in BPXYIPCP, and are mapped onto S\_TYPE, which is in BPXYMODE.

The values that begin with an "S\_I" prefix are defined in BPXYMODE, and are a subset of the access permissions that apply to files.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgget service returns -1 or the message queue identifier.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgget service stores the return code. The msgget service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The msgget service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	A message queue identifier exists for the Key parameter, but operation permission, as specified by the low-order 9– bits of the Message_Flag parameter, is not granted (the "S_" items). The following reason code can accompany the return code: JRIpcDenied.
<b>EEXIST</b>	A message queue identifier exists for the Key parameter, and both Ipc_CREAT and Ipc_EXCL are specified. The following reason code can accompany the return code: JRIpcExists.
<b>EINVAL</b>	The Message_Flag operand included bits that are not supported by this function. The following reason code can accompany the return code: JRIpcBadFlags.
<b>ENOENT</b>	A message queue identifier does not exist for the Key parameter, and Ipc_CREAT was not set. The following reason code can accompany the return code: JRIpcNoExist.
<b>ENOSPC</b>	The system limit on the number of message queue IDs has been reached. The following reason code can accompany the return code: JRIpcMaxIDs.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgget service stores the reason code. The msgget service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. As long as a thread knows the message queue ID, it may issue a msgctl, msgsnd, or msgrcv request (msgget is not needed).
2. This function returns the message queue identifier that is associated with the Key parameter.
3. This function creates a data structure that is defined by MSQID\_DS if one of the following is true:
  - The Key parameter is equal to Ipc\_PRIVATE.
  - The Key parameter does not already have a message queue identifier associated with it, and Ipc\_CREAT is set.
4. Upon creation, the data structure that is associated with the new message queue identifier is initialized as follows:
  - Ipc\_CUID and Ipc\_UID are set to the effective user ID of the calling task.
  - Ipc\_CGID and Ipc\_GID are set to the effective group ID of the calling task.
  - The low-order 9-bits of Ipc\_MODE are equal to the low-order 9-bits of the Message\_Flag parameter.
  - MSG\_QBYTES is set to the system limit that is defined by parmlib.
5. The message queue is removed from the system when msgctl is called with command Ipc\_RMID.
6. Users of message queues are responsible for removing them when they are no longer needed. Failure to do so ties up system resources.
7. In a client/server environment, two message queues could be used: one inbound to the server, created with Ipc\_SndTypePID, and the other outbound from the server, created with Ipc\_RcvTypePID. This arrangement guarantees that the server knows the process ID of the client

and that the client is the only process that receives the server's returned message. The server could call the msgrcv service with PID=0 to see if there are any messages that belong to process IDs that have gone away.

8. Message\_Flags Ipc\_PLO1 and Ipc\_PLO2 are ignored if the PLO (Perform Lock Operation) instruction is not present on the hardware. (See SCCBPLO in IHASCCB and the Ipc\_PLOInUse bit in the S\_MODE byte returned with w\_getipc.)
9. Performance of the PLO instruction for serialization varies with the msgrcv() type, the number of messages on the queue, and the number of tasks that are doing msgsnd and msgrcv requests. A msgrcv request with a message type that is less than zero and that has long message queues is expected to be a poor performer. A msgrcv request with a message type that is greater than zero is expected to be an equivalent or good performer. A msgrcv request with a message type equal to zero is expected to be a very good performer.
10. Message queues that are created with Ipc\_RcvTypePID, Ipc\_SndTypePID, Ipc\_PLO1 and Ipc\_PLO2 show these bits, and may show the Ipc\_PLOInUse bit in the S\_MODE byte that is returned with the w\_getipc request.
11. Message queue PLO serialization is not compatible with the use of select() for message queues. When the msgrcv service detects a select() for a message queue, serialization is changed to use traditional latches.
12. Performance runs should be made with Ipc\_PLO1, because Ipc\_PLO2 could switch to latch serialization, and the user would not be aware of this. Upon the first msgrcv() with a message type that is less than zero, the message queue will attempt to switch to latch serialization.

## Related services

- [“msgctl \(BPX1QCT, BPX4QCT\) — Perform message queue control operations” on page 455](#)
- [“msgrcv \(BPX1QRC, BPX4QRC\) — Receive from a message queue ” on page 462](#)
- [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue ” on page 465](#)
- [“w\\_getipc \(BPX1GET, BPX4GET\) — Query interprocess communications” on page 991](#)

## Characteristics and restrictions

1. There is a maximum number of message queues that are allowed in the system.
2. The invoker is restricted by ownership, read, and read-write permissions that are defined by msgget and msgctl Ipc\_SET.

## Examples

See [“BPX1QGT \(msgget\) example” on page 1276](#).

# msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue

## Function

The msgrcv service receives messages from a message queue.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1QRC):</b>	31-bit

Operation	Environment
<b>AMODE (BPX4QRC):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1QRC, (Message_Queue_ID,
               Message_Address,
               Message_Alet,
               Message_Length,
               Message_Type,
               Message_Flag,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4QRC with the same parameters. The `Message_Address` and `Message_Type` parameters are doublewords.

## Parameters

### Message\_Queue\_ID

Specifies the message queue identifier. The parameter must be of type *integer* and length *fullword*.

### Message\_Address

Specifies the name of a fullword (doubleword) field that contains the address of a buffer that is mapped by MSGBUF or MSGXBUF (see [“BPXYMSG — Map interprocess communication message queues” on page 1107](#)). The parameter must be of type *address* and length *fullword* (doubleword).

The `Message_Address` value must be greater than 0.

### Message\_Alet

The name of the fullword that contains the access list entry token (ALET) for `Message_Address`, which identifies the address space or data space where the buffer resides. The parameter must be of type *address* and length *fullword*.

- Specify a `Message_Alet` of 0 if the buffer is in the user's address space (the current primary address space).
- Specify a `Message_Alet` of 2 if the buffer is in the home address space.

If a value other than 0 or 2 is specified for the `Message_Alet`, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### Message\_Length

Specifies the length of the message text to be placed in the buffer that is pointed to by `Message_Address` parameter. The parameter must be of type *integer* and length *fullword*.

- In 31-bit mode, if `Msg_Info` is specified, this buffer is 20 bytes longer than `Message_Length`; otherwise, it is 4 bytes longer than `Message_Length`.
- In 64-bit mode, if `Msg_Info` is specified, this buffer is 32 bytes longer than `Message_Length`; otherwise, it is 8 bytes longer than `Message_Length`.

The message that is received might be truncated (see `MSG_NOERROR` of `Message_Flag`). A value of zero with `MSG_NOERROR` is useful for receiving the message type without the message text.

**Message\_Type**

Specifies the type of message that was requested, as follows:

- If Message\_Type is equal to zero, the first message on the queue is received.
- If Message\_Type is greater than zero, the first message of Message\_Type is received.
- If Message\_Type is less than zero, the first message of the lowest type that is less than or equal to the absolute value of Message\_Type is received.

The parameter must be of type *integer* and length *fullword* (doubleword).

**Message\_Flag**

The supplied parameter must be of type *integer* and length *fullword*.

MSG\_NOERROR specifies that the received message is to be truncated to Message\_Length (mapped in BPXYMSG). The truncated part of the message is lost, and no indication of the truncation is given to the caller.

MSG\_INFO specifies that the received message is to be of the MSGXBUF and not the MSGBUF format, mapped in BPXYMSG. MSG\_INFO specifies that extended information is to be received. This is similar to the msgxrcv() C language function.

IPC\_NOWAIT specifies the action that is to be taken if a message of the wanted type is not on the queue, as follows:

- If IPC\_NOWAIT is specified, the caller is to return immediately with an error (ENOMSG).
- If IPC\_NOWAIT is not specified, the calling thread is to suspend execution until one of the following occurs:
  - A message of the type is placed on the queue.
  - The message queue is removed from the system (EIDRM).
  - The caller receives a signal (EINTR).

MSG\_INFO\_EX specifies that the received message has the same format as the 64-bit MSGXBUF format, regardless of the caller's AMODE. That is, MSGX\_MTIME and MSGX\_TYPE are always 8 bytes.

**Return\_value**

The name of a fullword in which the msgrcv service returns -1, or the number of MSG\_MTEXT bytes returned. The returned parameter is of type *integer* and length *fullword*.

**Return\_code**

The name of a fullword in which the msgrcv service stores the return code. The returned parameter is of type *integer* and length *fullword*.

The msgrcv service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The msgrcv service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>E2BIG</b>	MSG_MTEXT is greater than Message_Length, and MSG_NOERROR is not set. The following reason code can accompany the return code: JRMSq2Big.
<b>EACCES</b>	Operation permission is denied to the calling task: JRIPcDenied. If the message queue was built with the Ipc_RcvTypePID, and the MSG_TYPE was other than the invoker's process ID, the following reason code accompanies the return code: JRTypeNotPID.
<b>EIDRM</b>	The Message_Queue_ID was removed from the system while the invoker was waiting. The following reason code can accompany the return code: JRIPcRemoved.
<b>EINTR</b>	The function was interrupted by a signal. The following reason code can accompany the return code: JRIPcSignaled.

Return_code	Explanation
<b>EINVAL</b>	Message_Queue_ID is not a valid message queue identifier; or the Message_Length parameter is less than 0 or the Message_Address is 0. The following reason codes can accompany the return code: JRipcBadID, JRMsqBadSize, or JrBadAddress.
<b>EFAULT</b>	The Message_Address parameter specified an address that caused the syscall to program check. The following reason code can accompany the return code: JRBadAddress.
<b>ENOMSG</b>	The queue does not contain a message of the desired type and Ipc_NOWAIT is set. The following reason code can accompany the return code: JRMsqNoMsg.

**Reason\_code**

The name of a fullword in which the msgrcv service stores the reason code. The returned parameter is of type *integer* and length *fullword*.

The msgrcv service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Within the type specifications, the longest waiting thread is reactivated first (FIFO). For example, if there are two threads waiting on message type 3 and one thread waiting on message type 2, when a message send for type 3 occurs, the oldest waiter for message type 3 receive is posted first.
2. Read access to the specified message queue is required.

**Related services**

- [“msgctl \(BPX1QCT, BPX4QCT\) — Perform message queue control operations” on page 455](#)
- [“msgget \(BPX1QGT, BPX4QGT\) — Create or find a message queue” on page 458](#)
- [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue ” on page 465](#)

**Characteristics and restrictions**

The caller of the msgrcv service is restricted by ownership, read, and read/write permissions that are defined by msgget and msgctl Ipc\_SET.

**Examples**

See [“BPX1QRC \(msgrcv\) example” on page 1277](#).

## **msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue**

---

**Function**

The msgsnd service sends a message to a message queue.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB

**Operation****Cross memory mode:****AMODE (BPX1QSN):****AMODE (BPX4QSN):****ASC mode:****Interrupt status:****Locks:****Control parameters:****Environment**

PASN = HASN

31-bit task or SRB mode

64-bit task mode only

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1QSN, (Message_Queue_ID,
               Message_address,
               Message_Alet,
               Message_Size,
               Message_Flag,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4QSN with the same parameters. The Message\_address parameter is a doubleword.

**Parameters****Message\_Queue\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

Specifies the message queue identifier.

**Message\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the message to be sent. The message\_address value must be greater than 0. This area is mapped by MSGBUF or MSGXBUF. (See [“BPXYMODE — Map the mode constants”](#) on page 1106.) The message type is the first word of the message and the message type must be greater than zero.

**Message\_Alet**

Supplied parameter

**Type:**

Address

**Length:**

Fullword



The name of the fullword that contains the ALET for Message\_address that identifies the address space or data space where the buffer resides.

Specify a Message\_Alet of 0 if the buffer is in the user's address space (current primary address space). Specify a Message\_Alet of 2 if the buffer is in the home address space.

If a value other than 0 or 2 is specified for the Message\_Alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### Message\_Size

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Specifies the length of the message text that is pointed to by the Message\_address parameter. The length does not include the 4-byte type that precedes the message text. For example, a message with a MSG\_TYPE and no MSG\_MTEXT would have a Message\_Size of zero. A zero-length message is accepted.

### Message\_Flag

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Specifies the action that is to be taken if one or more of these conditions are true:

- Placing the message on the message queue would cause the current number of bytes on the message queue (msg\_cbytes) to be greater than the maximum number of bytes that are allowed on the message queue (msg\_qbytes).
- The total number of messages on the message queue (msg\_qnum) is equal to the system-imposed limit.

The actions to be taken are as follows:

- If Ipc\_NOWAIT is specified, the caller returns immediately with an error (EAGAIN).
- If Ipc\_NOWAIT is not specified, the calling thread suspends execution until one of the following occurs:
  - The message is sent.
  - The message queue is removed from the system (EIDRM).
  - The caller receives a signal (EINTR).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the msgsnd service returns -1 or 0. The message was sent unless a -1 is received.

### Return\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the msgsnd service stores the return code. The msgsnd service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The msgsnd service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Operation permission is denied to the calling task: JRIPcDenied. If the message queue was built with Ipc_SndTypePID, and the MSG_TYPE was other than the invoker's process ID, the following reason code accompanies the return code: JRTypeNotPID.
<b>EAGAIN</b>	The message cannot be sent, and Message_Flag is set to Ipc_NOWAIT. The following reason codes can accompany the return code: JRMSqQueueFullMessages, JRMSqQueueFullBytes.
<b>EIDRM</b>	The Message_Queue_ID was removed from the system while the caller was waiting. The following reason code can accompany the return code: JRIPcRemoved.
<b>EINTR</b>	The function was interrupted by a signal, and the message was not sent. The following reason code can accompany the return code: JRIPcSignaled.
<b>EINVAL</b>	Message_Queue_ID is not a valid message queue identifier; the value of MSG_TYPE is less than 1; or the value of Message_Size is less than zero or greater than the system-imposed limit. or the Message_address is 0. The following reason codes can accompany the return code: JRIPcBadID, JRMSqBadSize, JRMSqBadType, or JrBadAddress.
<b>EFAULT</b>	The Message_address parameter specified an address that caused the service to program check. The following reason code can accompany the return code: JRBadAddress.
<b>ENOMEM</b>	The message was not sent due to insufficient system storage exits. The following reason code can accompany the return code: JrSmNoStorage.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the msgsnd service stores the reason code. The msgsnd service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

- Write access to the specified message queue is required.
- For more information about programming considerations for the SRB mode, see [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041.

**Related services**

- [“msgctl \(BPX1QCT, BPX4QCT\) — Perform message queue control operations”](#) on page 455
- [“msgget \(BPX1QGT, BPX4QGT\) — Create or find a message queue”](#) on page 458
- [“msgrcv \(BPX1QRC, BPX4QRC\) — Receive from a message queue ”](#) on page 462

## Characteristics and restrictions

The caller of this service is restricted by ownership and read and read/write permissions that are defined by msgget and msgctl Ipc\_SET.

### Examples

See “BPX1QSN (msgsnd) example” on page 1277.

## msync (BPX1MSY, BPX4MSY) – Synchronize memory with physical storage

---

### Function

The msync callable service writes all modified pages over the requested range to their permanent storage locations on disk. It also deletes any in-memory cached pages over the requested range, resetting the contents of those pages to that which resides on disk.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, PSW Key 2 or PSW Key 8
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MSY)</b>	31-bit
<b>AMODE (BPX4MSY)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1MSY, (Map_address,
               Map_length,
               Sync_Options,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MSY with the same parameters. The Map\_address and Map\_length parameters are doublewords.

### Parameters

#### Map\_address

Supplied parameter.

#### Type

Address

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the mapping from which the modified pages are to be written to their permanent storage locations on disk, or invalidated. The value of `map_address` must be a multiple of the page size.

**BPX1MPR only:** If the high-order bit is on, the other 31 bits of the fullword parameter are interpreted as a pointer to a doubleword location. This doubleword is used as the 64-bit map address and must not be in the range 2 G to 64 G.

**BPX4MSY only:** The doubleword parameter allows a nonzero value in the high-order fullword. The map address must not be in the range 2 G to 64 G.

**Map\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword (doubleword)

The name of the fullword (doubleword) that contains the size (in bytes) of the mapping that is to have all updated pages written out to disk, or invalidated. The length can be the size of the whole mapping, or a part of it. If the specified length is not a multiple of the page size, it is rounded up to a page boundary.

**BPX1MPR only:** If the high order bit is on, the other 31 bits of the fullword parameter will be interpreted as a pointer to a doubleword location. This doubleword will be used as the 64-bit Map Address and the first fullword must be nonzero.

**BPX4MSY only:** The doubleword parameter will allow a nonzero value in the high-order fullword.

**Sync\_Options**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of the fullword that contains the option flags for the service. The specified value can be a combination (for example, using an exclusive OR) of one or more of the following flags, with the limitation that `MS_ASYNC` and `MS_SYNC` are mutually exclusive. These constants are defined in the `BPXYCONS` macro.

Constant	Description
<b>MS_ASYNC</b>	Performs asynchronous writes. <code>MS_ASYNC</code> returns immediately when all write operations are scheduled. You want to write consistent data to the disk, do not use this option.
<b>MS_SYNC</b>	Performs synchronous writes. <code>MS_SYNC</code> will return after all write operations are completed.
<b>MS_INVALIDATE</b>	Invalidates the cached memory-mapped pages. After the cached copy of the data in memory has been invalidated for a <code>MAP_SHARED</code> mapping, any further references to these pages will be obtained by the system from their permanent storage locations on disk. For a <code>MAP_PRIVATE</code> mapping, only updated (private) pages are invalidated. Any further references to these pages will be obtained from the shared cache.

**Notes:**

1. If MS\_INVALIDATE is the only flag that is specified, the requested cached memory—mapped pages are invalidated without any modified pages first being written to disk.
2. If MS\_INVALIDATE is specified with either MS\_SYNC or MS\_ASYNC, all the modified pages in the requested address range are written to disk before the cached copy of data in memory is invalidated.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the msync service returns the value of 0 if the request is successful, or -1 if it is not successful.

Upon successful completion, the msync service writes all modified pages over the range (map\_address, map\_address + map\_length) to their permanent storage locations on disk, invalidates the cached mmap pages, or does both.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the msync service stores the return code. The msync service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The msync service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)
<b>EINVAL</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• The value of map_address is not a multiple of the page size. (JRNotPage).</li> <li>• The value in the Sync_Options parameter is incorrect. (JROptNotSupp).</li> <li>• The input address or length is negative. (JRNegativeValueInvalid).</li> <li>• The caller's PSW key does not match the key of the memory mapped storage segment that is being operated against. (JrKeyMismatch).</li> <li>• In 64-bit mode, an address greater than 31-bit addr was passed in map_address (JrAddressNotAvailable).</li> <li>• In 64-bit mode, a length greater than X'FFFFFFFF' was passed in map_length (JrInvParmLength).</li> </ul>
<b>EIO</b>	An I/O error occurred during the write to the file system (file system JR). This return code is set only if MS_SYNC is set in the Sync_Options parameter. I/O errors during asynchronous write operations are not reported to the application.

**Return\_code****ENOMEM****Explanation**

One of the following conditions occurred:

- Some or all of the addresses in the range (map\_address, map\_address + map\_length) are not valid for the address space. (JRAddressNotAvailable)
- One or more specified pages are not mapped. (JRNotMapped)

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the msync service stores the reason code. The msync service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The msync service is used by programs that require a file to be in a known state (such as in building transaction-oriented programs).
2. When a request is made to write the cached pages by a process that has mapped the area with the MAP\_SHARED option, updates made by all processes sharing the specified file-fset range are written, not just the updates made by the msync requesting process. The same is true for invalidate requests.
3. Only full pages are processed. If the map\_length parameter contains a value that is not a multiple of the page size, the length is rounded up to a full page.
4. In relation to advisory locking mechanisms, there is no difference between sharing a file by using the mmap services, and sharing a file by using the read/write services. Specifically, before a series of bytes is accessed by using either method, a byte range lock is required to ensure the consistency of the data being accessed. If the intent is to write consistent data to the disk when a file is shared by using memory map services, an advisory lock should be held on the pages being acted upon, before calling the msync service with the MS\_SYNC option.
5. Constants that are used for this callable service are defined in the BPXYCONS macro. See [“BPXYCONS — Constants used by services”](#) on page 1062.

**Related services**

- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ”](#) on page 913
- [“mmap \(BPX1MMP, BPX4MMP\) — Map pages of memory ”](#) on page 435
- [“mprotect \(BPX1MPR, BPX4MPR\) — Set protection of memory mapping ”](#) on page 452
- [“munmap \(BPX1MUN, BPX4MUN\)— Unmap previously mapped addresses ”](#) on page 473

**Characteristics and restrictions**

1. The range that is specified (map\_address, map\_address + map\_length) must not contain any areas that are not currently memory mapped. However, it may contain areas that were unmapped, in which case no action is taken against the unmapped areas.
2. To write or invalidate MAP\_SHARED mappings, the range that is specified must have the PROT\_WRITE access level. If any portion of the specified range has either the PROT\_NONE or PROT\_READ access levels at the time of the msync request, that portion will not be written or invalidated, and no error condition will be raised.

3. Because memory map is implemented using a cached copy of the original data that resides on disk, concurrent updates that are made by using the write callable service to a file that is being memory-mapped will produce undefined results. If you want this type of activity, then explicit serialization must be implemented between a process invoking the msync service with the invalidate option, and another process invoking the write service (page-multiple advisory lock).
4. When the msync service is called for MAP\_PRIVATE mappings, any data that is modified by that process is not written to the file, and such data is not visible to other processes. The only supported action is to invalidate the pages that were cached exclusively for the use of the requesting process (this has no impact on the MAP\_SHARED cache). For the invalidate request to be successful, the range that is specified must have the PROT\_WRITE access level. If another process mapping the same file-offset range with the MAP\_SHARED option invalidates the shared cache, then, from the perspective of the MAP\_PRIVATE process, only the pages that were not updated by the MAP\_PRIVATE process (still shared) are invalidated. The modified (and now private) pages remain intact in the cache. This type of activity could cause inconsistencies within the MAP\_PRIVATE mapping.

### Examples

See [“BPX1MSY \(msync\) example”](#) on page 1266.

## munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses

---

### Function

The munmap callable service removes the mapping for pages in the requested range. Only use it to unmap regions that were previously mapped by the application with the mmap callable service.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, PSW Key 2, or PSW Key 8
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1MUN)</b>	31-bit
<b>AMODE (BPX4MUN)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1MUN, (Map_address,
               Map_length,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MUN with the same parameters. The Map\_address and Map\_length parameters are doublewords.

## Parameters

### Map\_address

Supplied parameter.

#### Type

Address.

#### Length

Fullword (doubleword).

The name of a fullword (doubleword) that contains the address of an existing mapping that is to be unmapped. The specified address does not have to be the start of a mapping. The value of map\_address must be a multiple of the page size. If the address falls within a MAP\_MEGA map, the address that is provided is rounded down to a megabyte multiple so that an entire segment is included in the unmap operation. You cannot unmap part of a segment when a MAP\_MEGA map is being processed.

**BPX1MUN only:** If the high-order bit is on, the other 31 bits of the fullword parameter is interpreted as a pointer to a doubleword location. This doubleword is used as the 64-bit map address.

**BPX4MUN only:** The doubleword parameter allows a nonzero value in the high-order fullword.

### Map\_length

Supplied parameter.

#### Type

Integer.

#### Length

Fullword (doubleword).

The name of the fullword (doubleword) containing the size (in bytes) of the mappings that are to be unmapped. The length can be the size of the whole mapping, or a part of it. If the specified length is not in multiples of the page size, it is rounded up to a page boundary. If map\_address plus map\_length falls within a MAP\_MEGA map, the length is rounded up so that it includes an entire segment (but not necessarily the entire MAP\_MEGA mapping).

**BPX1MUN only:** If the high-order bit is on, the other 31 bits of the fullword parameter is interpreted as a pointer to a doubleword location. This doubleword is used as the 64-bit Map length.

**BPX4MUN only:** The doubleword parameter allows a nonzero value in the high-order fullword.

### Return\_value

Returned parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword in which the munmap service returns the value of 0 if the request is successful, or -1 if it is not successful.

Upon successful completion, the munmap service unmaps all pages in the range (map\_address, map\_address + map\_length).

### Return\_code

Returned parameter.

#### Type

Integer.

#### Length

Fullword.



The name of a fullword in which the munmap service stores the return code. The munmap service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*.

The munmap service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EAGAIN</b>	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)
<b>EINVAL</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• The value of <code>map_address</code> is not multiples of the page size. (JRNotPage).</li> <li>• Some or all of the addresses in the range (<code>map_address</code>, <code>map_address + map_length</code>) are not valid for the address space. (JRAddressNotAvailable).</li> <li>• The input address is negative, or the input length is zero or negative. (JRZeroOrNegative).</li> <li>• The caller's PSW key does not match the key of the memory-mapped storage segment that is being unmapped. (JrKeyMismatch).</li> <li>• In 64-bit mode, an address greater than 31-bit addr was passed in <code>map_address</code> (JrAddressNotAvailable).</li> <li>• In 64-bit mode, a length greater than X'7FFFFFFF' was passed in <code>map_length</code> (JrInvParmLength).</li> </ul>

#### **Reason\_code**

Returned parameter.

#### **Type**

Integer.

#### **Length**

Fullword.

The name of a fullword in which the munmap service stores the reason code. The munmap service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## **Usage notes**

1. An address of 0 with a length of X'7FFFF000' unmaps all the storage that is associated with this process.
2. For both private and shared mappings, if the munmap service unmaps a subset of the range of the original mmap request, further references to those pages result in a program check exception. When the entire range of the original mmap request has been unmapped, the memory that is allocated by the mmap service is freed.
3. If there are no mappings in the requested address range, the munmap service has no effect. The service returns successfully.
4. The range that is specified (`map_address`, `map_address + map_length`) may contain areas that have been unmapped, in which case no action is taken against the unmapped areas.
5. If a mapping to be removed is private, any modifications that are made in the specified address range are discarded.
6. If a mapping to be removed is shared, all modifications that are made in the specified address range since the last msync (if any) are written to disk. If this is not desired, the msync service must be called to invalidate the updates that have been made to the mapped region before the range is unmapped.

7. If a memory-mapped region is not unmapped before the process terminates, process termination does not automatically write out to disk any modified data in the mapped region. Modified private data in a MAP\_PRIVATE region is discarded. If the mapped region is MAP\_SHARED, the modified data continues to reside in the cache (if the same file-offset range is being shared), and may ultimately be written out to disk by another process via the msync service. However, if no other processes map the same file-offset range as MAP\_SHARED, the modified data is discarded.
8. Only entire pages are unmapped. If the map\_length parameter contains a value that is not a multiple of the page size, the length is rounded up to a full page. For MAP\_MEGA maps, only entire segments are unmapped. The map\_address and map\_length are adjusted to ensure that entire segments are unmapped.
9. An unmap request may span MAP\_MEGA and non-MAP\_MEGA ranges.

**Related services**

- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ” on page 913](#)
- [“mmap \(BPX1MMP, BPX4MMP\) — Map pages of memory ” on page 435](#)
- [“mprotect \(BPX1MPR, BPX4MPR\) — Set protection of memory mapping ” on page 452](#)
- [“msync \(BPX1MSY, BPX4MSY\) — Synchronize memory with physical storage ” on page 469](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1MUN \(munmap\) example” on page 1266](#).

## **mvsiptaffinity (BPX1IPT, BPX4IPT) — Run a program on the IPT thread**

---

**Function**

The mvsiptaffinity callable service allows a task created with pthread\_create to request that a user-defined assembler routine run on its initial pthread-creating thread (IPT). The requesting pthread is blocked until the requested routine has been executed.

This service manages MVS resources under the IPT, instead of under the task created with pthread\_create. Some resources that can be managed with this service are:

- Load modules
- Opened data sets
- Other MVS resources with task affinity

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1IPT):</b>	31-bit
<b>AMODE (BPX4IPT):</b>	64-bit
<b>ASC mode:</b>	Primary mode

<b>Operation</b>	<b>Environment</b>
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1IPT, (Routine_address,
               Parameter_list,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4IPT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Routine\_address

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the routine control is passed to on the pthread's IPT. The requesting pthread is responsible for ensuring that the routine to be run is in memory when it is called and remains there until the call is complete.

### Parameter\_list

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the routine parameter list. The value in this fullword is passed in register 1 when the specified routine receives control. If the routine does not require parameters, specify 0.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the mvsiptaffinity service returns -1 if it is not successful. If it is successful, the mvsiptaffinity service returns 0.

### Return\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the bvsiptaffinity service stores the return code. The bvsiptaffinity service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The bvsiptaffinity service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	A bad address was received as an argument of the call, or the specified routine experienced an abend or program check that was not handled by the routines recovery. The following reason codes can accompany the return code: JRBadAddress and JRRoutineError.
<b>EAGAIN</b>	Another pthread within the process has this call pending. At most one pthread can request this service at a time. The requesting task can try again later when the current pending call is complete.
<b>EACCES</b>	A task other than a pthread-created task is not permitted to perform this service.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the bvsiptaffinity service stores the reason code. The bvsiptaffinity service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

- The specified routine to be executed receives control with the following attributes:
  - Problem program state
  - Key of the IPT task
  - AMODE 31(64), according to the AMODE of the caller
  - Primary ASC mode
- The register usage on entry to the specified routine is:
  - R0: Undefined
  - R1: Address of Parameter\_list, as specified by the caller of the bvsiptaffinity service
  - R2–R12: Undefined
  - R13: Address of a 72-byte work area with which the routine gains control. For AMODE 64 callers, the work area is 144 bytes long.
  - R14: The return address from the specified routine to the bvsiptaffinity service. This address must be preserved by the invoked routine.
  - R15: Address of the invoked routine
- Only tasks created with pthread\_create can invoke this service. If a task that was created using MVS non-POSIX interfaces requests this service, or if it is an IPT itself, it receives an EACCES return code.
- At most one pthread can have this service request pending at a time. If a pthread already has this service pending, when another pthread requests this service, the last pthread receives an EAGAIN

return code. It is the caller's responsibility to serialize the invocation of mvspiaffinity, or contain retry logic if the EAGAIN return code is obtained.

5. The EXITRTN assembler routine cannot issue callable services after it gains control under the IPT.
6. The specified routine can establish its own recovery environment. However, even if recovery is not established, the mvspiaffinity service establishes its own recovery environment while running under the IPT. For all recoverable errors, this recovery routine retries, returning the EFAULT return code to the requestor. It also ensures that any recovery routine established by the IPT itself is not entered unexpectedly.

## Related services

There are no related services.

## Characteristics and restrictions

None.

## Examples

See [“BPX1IPT \(mvspiaffinity\) example” on page 1253](#).

# mvspause (BPX1MP, BPX4MP) – Wait on user events plus signals

## Function

The mvspause callable service allows a thread to suspend until a signal arrives or some application-defined event is posted.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task - No EUT FRRs
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MP):</b>	31-bit
<b>AMODE (BPX4MP):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MP,( Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MP.

## Parameters

### Return\_Value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service returns a 0 indicating that an event occurred, or -1 otherwise.

### Return\_Code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvspause service stores the return code. The mvspause service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mvspause service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	Error addressing parameters. The parameters on the prior mvspauseinit call were not fully validated at mvspauseinit time. The following reason code unique to the mvspause service can accompany the return code: JRECBStateBad.
<b>EINTR</b>	The mvspause call was interrupted by a signal.
<b>EMVSPARM</b>	Incorrect parameters were passed to an MVS service. The following reason codes unique to the mvspause service can accompany the return code: JRECBListNotSetup, JRECBStateBad.

### Reason\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the service routine stores the reason code. The reason code further qualifies the return code value. The mvspause service stores a reason code only when the return value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The intended use of mvspause is for a program to wait on user ECBs plus signals.
2. The user must first invoke the mvspauseinit service to declare to the system the list of ECB pointers to use. The system makes a copy of the list of ECB pointers to wait on and uses the existing MVS WAIT service to wait for the user events and the signal event.
3. When one of the ECBs in the ECB list has been posted or a signal is received, the mvspause operation concludes and control is returned to the caller. When a signal is received, the mvspause service posts the signal ECB and runs the signal handler before returning to the user.

4. The user has the option of reinvoking the mvspause service without reinvoking the mvspauseinit service. The user should be aware, however, that the system has made a copy of the list of pointers that point to the user's ECBs. Any changes to the caller's copy of the ECB pointer array are not reflected in the system copy unless the mvspauseinit service is invoked again. Furthermore, when the user wishes to reinvoke the mvspause without reinvoking the mvspauseinit service, the user must clear all ECBs that were posted. This includes clearing the signal ECB. If the user does not clear posted ECBs, the mvspause detects the already posted ECB and returns immediately. The user must take care when clearing ECBs, because not all ECBs may have been posted. Asynchronous operations could post an ECB at any time during the user's processing. The user should clear only ECBs that are processed, and not blindly clear all ECBs.

The following logic example displays one method for processing ECBs:

```
CALL MVSpauseInit(list of ECB addresses);

for(;;)          /* Do forever */
{
    call MVSpause()
    for(i=1;i<=MaxEcb;i++)
    {
        Copy value of Ecb(i)
        if (Ecb(i) was posted)
        {
            Clear Ecb(i)
            switch(i)
            {
                case 1:    CALL Signal-Occurred;
                           break;
                case 2:    CALL Rtn_for_2nd_Ecb;
                           break;
                case 3:    CALL Rtn_for_3rd_Ecb;
                           break;
                ...        /* As many as are needed */
                default: no ECBs POSTed
            }
            /* end switch */
        }
        /* end if */
    }
    /* end for */
    if (terminating condition occurred)
        break;
    /* Exit Do Forever loop */
}
/* end do forever */
```

## Related services

- [“mvspauseinit \(BPX1MPI, BPX4MPI\) — Set up to wait on user events plus signals” on page 481](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1MP \(mvspause\) example” on page 1263](#).

# mvspauseinit (BPX1MPI, BPX4MPI) — Set up to wait on user events plus signals

## Function

The mvspauseinit callable service allows the thread to declare to the system a list of event control blocks (ECBs) the application program will use to receive event notifications. This service is used in conjunction with the mvspause service.

## Requirements

### Operation

#### Authorization:

Supervisor state or problem state, any PSW key

#### Dispatchable unit mode:

Task - No EUT FRRs

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1MPI):

31-bit

#### AMODE (BPX4MPI):

64-bit

#### ASC mode:

Primary address space control (ASC) mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MPI, ( Addr_of_ECBlst,
                Return_value,
                Return_code,
                Reason_code)
```

AMODE 64 callers use BPX4MPI with the same parameters. The Addr\_of\_ECBlst parameter is a doubleword.

## Parameters

### Addr\_of\_ECBlst

Address of a list of up to 1018 user-defined event control blocks (ECBs). The system uses the first ECB in the list.

#### Type:

Pointer

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) from which the service extracts the address of the input ECB list. The mvspauseinit service requires this list to contain a maximum of 1018 ECBs, with the first ECB dedicated to the system. The user is responsible for obtaining the storage for all ECBs.

All pointers in the ECB list are 32-bit pointers for both AMODE 31 and AMODE 64 callers, as ECBs are only supported below the 2 GB bar. The last ECB pointer in the list must have the high-order bit set to 1 (80000000x). This bit indicates that it is the last ECB address in the list.

### Return\_Value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the service returns a 0 upon normal completion, or -1 otherwise.

### Return\_Code

Returned parameter



**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvspauseinit service stores the return code. The mvspauseinit service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mvspauseinit service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EFAULT	Error addressing parameters. The following reason codes unique to the mvspauseinit service can accompany the return code: JRECBError, JRECBListBad.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

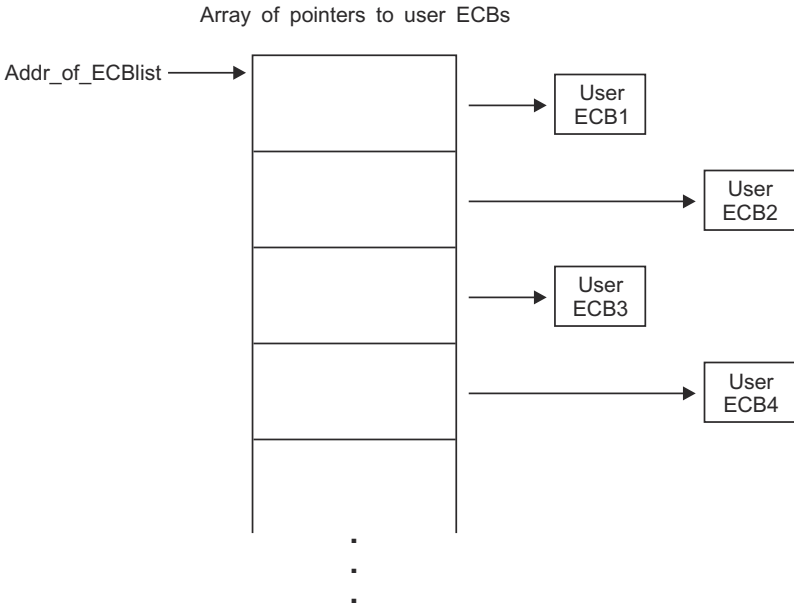
Fullword

The name of a fullword in which the service routine stores the reason code. The reason code further qualifies the return code value. The mvspauseinit service stores a reason code only when the return value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The intended use of the mvspauseinit service is for a program to declare to the system a list of pointers to user ECBs in user-managed storage. These ECBs are used by the mvspause function to suspend the thread until a signal arrives or a user-defined event is posted.
2. The user is responsible for initializing all ECBs, including the signal ECB. The first ECB is the signal ECB. The system does not alter the ECBs during mvspauseinit. This means that an asynchronous operation may post an ECB in the ECB list while mvspauseinit is operating.
3. After mvspauseinit returns to the caller, the mvspause service may be invoked as many times as necessary without reinvoking the mvspauseinit service. If the application program needs to change one or more ECB addresses, the application must reinvoke the mvspauseinit service before invoking the mvspause service.

**Note:** Only one ECB list is allowed per thread. If a user invokes the mvspauseinit service multiple times, each invocation replaces the ECB list specified on previous invocations of mvspauseinit.



**Related services**

- [“mvspause \(BPX1MP, BPX4MP\) – Wait on user events plus signals” on page 479](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1MPI \(mvspauseinit\) example” on page 1265](#).

**mvspocclp (BPX1MPC, BPX4MPC) – Clean up kernel resources**

**Function**

The mvspocclp callable service cleans up the z/OS UNIX-related resources for an entire process or on a thread-by-thread basis. After cleaning up resources, the mvspocclp service terminates the thread or the entire process with the final thread.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MPC):</b>	31-bit
<b>AMODE (BPX4MPC):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax diagram is as follows:

```
CALL BPX1MPC, (Status_field,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MPC with the same parameters.

**Parameters****Status\_field**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword status field. The status field is a one-word area that is mapped by BPXYWAST, the WAIT status word. The WAST area should be initialized to zero. If the caller wants to set a specific exit status, then either WASTEXITCODE or WASTSIGTERM should be set. If the invocation of this service causes a full process cleanup to occur and the contents of the status field conform to the allowable exit status values, the contents are made available to the parent when the wait service is issued. For the mapping of the status field and a description of the allowable exit status values see [“BPXYWAST — Map the wait status word”](#) on page 1172.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword to which the mvspocclp service returns one of the following values:

Value	Explanation
0	Thread-related resources were cleaned up for the calling thread.
1	Process-related resources were cleaned up for the calling process.
-1	The service failed to clean up process resources.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvspocclp service stores the return code. The mvspocclp service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return](#)

codes (errno) in *z/OS UNIX System Services Messages and Codes* The mvspocclp service can return the following value in the Return\_code parameter:

Return_code	Explanation
<b>EMVSERR</b>	The specified terminating status value did not conform to the allowable values. The call failed with a Return_value of -1 and a Reason_Code of JrInvTermStat.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword where the mvspocclp service stores the reason code. The mvspocclp service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see Reason codes in *z/OS UNIX System Services Messages and Codes*.

### Usage notes

- The mvspocclp service normally cleans up only the thread-related data for the calling thread. The two following situations, however, cause full process cleanup to occur:
  - If the call is made from the initial thread of the process and no other threads exist in the process.
  - If the call is made from the last thread that is left in the process and that thread is not the initial thread, and the initial thread has not performed any z/OS UNIX system calls.

In these two cases, both the thread-related and process-related resources are cleaned up and z/OS UNIX process termination is performed. See the \_exit service for a description of z/OS UNIX process ending.
- An important distinction between this service and the \_exit service is that this service does not actually cause the user's tasks to end. The \_exit service has the added effect of causing a full MVS-like ending, in that all the tasks in the executing process are ended. The mvspocclp service cleans up only the process-related resources and causes a process termination to occur, leaving the other MVS-related resources in the address space unaffected.
- The mvspocclp service does not trigger a core dump when the dump flag is on in the status word.
- For message queues, each thread is removed from the send and receive waiting chains (the message to be sent is lost). End of memory may require the message queue to be rebuilt.
- When shared memory is being used, each process is terminated and the shared memory segment attached to the terminating process is detached. If the last attachment is removed and a shmctl RMID had been issued, the segment is removed from the system.
- If semaphores are being used, each thread is removed from the waiting chain. The adjustment values are associated with the process, not the thread. The adjustments are made to each semaphore set atomically. If an adjustment would cause a semaphore value to overflow a limit (0 or SEM#MAX\_VAL), no adjustment is made to that semaphore. Adjustments will continue for the set. No assumptions may be made as to the order in which the semaphore sets will be adjusted.
 

When semval changes, the waiting chain is searched and other threads may regain control (as with semop, semctl operations). As adjustments are completed, sem\_pid and sem\_otime are updated for each semaphore set.
- Even if full process termination does not occur, mvspocclp will cause the terminating thread to wait up to 120 seconds for subtask termination. If the caller is the IPT, a reason code will be returned to indicate that a subtask is still attached.
- By default, when an mvspocclp call results in a full process cleanup, all processes that are found in the caller's subtask tree are terminated with a sigkill signal.

If the ThliUndubCallerOnly flag is on in the caller's THLI when the mvspocclp service is called, mvspocclp does not attempt to terminate processes in the caller's subtask tree.

You cannot use the ThliUndubCallerOnly flag to clean up the first process dubbed in the address space while other processes exist in the caller's address space. The mvspocclp service will fail with return code EMVSERR and reason code JRActiveProcess.

9. WASTEXITCODE is a two-byte field that can be set to any value from 1 to 255. This is considered a user specified value for successful termination.
10. WASTSIGTERM is a two-byte field that can be set to any valid signal number (defined in BPXYSIGH). If the signal specified is SIGKILL (x'09'), then the termination request is treated as an abnormal termination, as if the task was abnormally terminated.

## Related services

- [“\\_exit \(BPX1EXI, BPX4EXI\) — End a process and bypass the cleanup ” on page 180](#)
- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)

## Characteristics and restrictions

1. The mvspocclp service is provided for non-C applications that invoke z/OS UNIX services. As a rule-of-thumb, if your program causes the task to be dubbed, issue mvspocclp when it is complete. If your program is already dubbed when invoked, do not call mvspocclp when exiting. If you know the termination of your program will cause the task to terminate, you can allow end-of-task processing to perform mvspocclp for you. To determine if your program is already dubbed, you can test STCBOTCB in mapping macro IHASTCB. If this field is 0, it is not dubbed. You can also use the querydub callable service ([“querydub \(BPX1QDB, BPX4QDB\) — Obtain the dub status of the current task ” on page 646](#)).
2. If a thread issued an attach\_exec or an attach\_execmvs, mvspocclp can be called to allow up to 60 seconds for a full MVS subtask termination to complete. Although mvspocclp would not trigger the termination of a subtask, a thread that exited the system while subtasks were attached would force those subtasks to terminate with an abend.

## Examples

See [“BPX1MPC \(mvspocclp\) examples” on page 1264](#).

# mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals

## Function

The mvssigsetup callable service allows a task to catch or intercept signals. This service also allows a task to intercept cancellation and quiesce interrupts. Only one mvssigsetup service in a process can be active. If a second mvssigsetup service must be performed in a process, an mvsunsigsetup service must be performed on the thread that issued the mvssigsetup service request before the second invocation of the mvssigsetup service.

Both MVS task termination and the mvspocclp service (BPX1MPC, BPX4MPC) perform the mvsunsigsetup service.

## Requirements

Operation	Environment
<b>Authorization:</b>	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN

Operation	Environment
<b>AMODE (BPX1MSS):</b>	31-bit
<b>AMODE (BPX4MSS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MSS,(Signal_interface_routine_address,
               User_data,
               Default_override_signal_set,
               Default_terminate_signal_set,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MSS with the same parameters. The `Signal_interface_routine_address` parameter and the `User_data` parameter are doublewords.

## Parameters

### Signal\_interface\_routine\_address

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the user-supplied signal interface routine (SIR) that gets control when a signal handler needs to be invoked. The signal handler is defined by the sigaction call; see [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action” on page 834](#). You can also invoke the SIR to process a default signal action, depending on the values specified for `Default_override_signal_set`. See the usage note on using the upper bit of the SIR address for indirect signal handler addresses.

### User\_data

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Fullword (doubleword)

The name of a fullword (in 31-bit mode) or doubleword (in 64-bit mode) containing 4 or 8 bytes of user-supplied data that is passed to the signal interface routine on invocation from signal processing.

### Default\_override\_signal\_set

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

8 bytes

The name of an 8-byte area containing a 64-bit mask of signals that the SIR processes when their respective default actions take place. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. The signals SIGSTOP, SIGDUMP, and SIGTRACE cannot be intercepted. The bit positions that represent these signals are ignored. Signal 64 represents cancellation or quiesce requests. For more information, see [“BPXYSIGH — Signal constants” on page 1148](#).

**Default\_terminate\_signal\_set**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

8 bytes

The name of an 8-byte area containing a 64-bit mask of signals specified in the Default\_override\_signal\_set parameter that also causes the process to end. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. When set to 1, the signal represented results in a task that is either stopped or in a wait state to be interrupted by the signal. It is up to the signal interface routine to end the process. The bit that represents signal 64 of this mask is reserved. For more information, see [“BPXYSIGH — Signal constants” on page 1148](#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvssigsetup service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvssigsetup service stores the return code. The mvssigsetup service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mvssigsetup service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EMVINITIAL</b>	The service failed. The following reason codes can accompany the return code: JRNotPRB, JRPSWKeyNotValid, and JRAIreadySigSetup.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvssigsetup service stores the reason code. The mvssigsetup service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

The user-supplied signal interface routine (SIR) is given control when the current PSW key is equal to the signal delivery key for the process. The signal delivery key for the process is defined as the PSW key when the process was dubbed for the first request for a callable service. A process image that results after the exec service or the execmvs service always has a signal delivery key of 8 and is not set up for signals.

If the signal handler addresses specified on the call to sigaction are not the actual handler addresses but pointers to the handler addresses, turn on the upper bit of the SIR address supplied on this service to enable ptrace to set breakpoints at the beginning of the signal handlers.

For information about the BPXYPPSD macro, see [“BPXYPPSD — Map signal delivery data” on page 1124](#).

The SIR receives control with the following register interface:

Register	Contents
<b>Reg 0</b>	0
<b>Reg 1</b>	Address of standard parameter list. PARM1= address of BPXYPPSD; Reg 1 = ADDR(PpsdSirPARMS).
<b>Regs 2–12</b>	0
<b>Reg 13</b>	0 No save area for registers is provided to the SIR. The SIR does not save caller's registers.
<b>Reg 14</b>	0 No return address.
<b>Reg 15</b>	Set to address of the SIR.

The SIR receives control in the following system state:

Operation	Environment
<b>Authorization:</b>	Problem program state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE:</b>	The same as the caller of mvssigsetup (BPX1MSS, BPX4MSS)
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Signal mask:</b>	All signals that may be blocked by the signal mask are blocked.

Following are the steps that a user-supplied SIR must perform.

1. The SIR must obtain local storage for a local copy of the BPXYPPSD and copy the BPXYPPSD information into this local storage.



2. The PPSD contains the information necessary for the SIR to determine the reason for the interruption. The interruption can be the result of a signal, cancellation, or quiesce request.
3. If the interrupt cannot be processed at this time, possibly due to general register 13 not currently containing the address of a program stack, or the last service called on the current thread was `cond_setup`, then the `queue_interrupt` service request is issued (see [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered”](#) on page 648). Then go to step “11” on page 491.
4. If the interrupt is a signal and the default action is to be performed by the SIR, write the appropriate messages to the terminal and end the process. For more information about how to end the process, see [“\\_exit \(BPX1EXI, BPX4EXI\) — End a process and bypass the cleanup”](#) on page 180.
5. If the interrupt is a cancellation or a terminating quiesce request, clean up any necessary thread-related resources and end the thread. To end the thread, issue the `pthread_get_and_exit` service with `Options_field` set to `PTEXTITTHREAD`. If the interrupt is because of a cancellation, issue the `pthread_exit_and_get` service with `Status_field` set to `-1`. For more information about how to end the thread, see [“pthread\\_exit\\_and\\_get \(BPX1PTX, BPX4PTX\) — Exit and get a new thread”](#) on page 588. If the interrupt is a freeze quiesce request, issue the `quiesce_threads` service to freeze the thread; see [“pthread\\_quiesce \(BPX1PTQ, BPX4PTQ\) — Quiesce threads in a process”](#) on page 598. The SIR receives these types of interrupts only if bit 64 of the `Default_override_signal_set` is set on.
6. Obtain language stack storage for the signal handler.
7. Examine the `sigaction` call flags in the `BPXYPPSD` for the signal being delivered. Some of these flags, specified on the `sigaction` call, are intended to allow the user certain options when interfacing with signal catchers, or to provide additional processing. For example, the `SA_SIGINFO` flag specifies that additional signal information, also present in `BPXYPPSD`, should be passed to the signal catcher in a `siginfo` structure. It is up to the SIR to interpret and implement these `sigaction` flags. Refer to [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action”](#) on page 834 for more information about the function of the `sigaction` flags. The relevant flags are:
  - `SA_ONSTACK`
  - `SA_RESETHAND`
  - `SA_RESTART`
  - `SA_SIGINFO`
  - `SA_NODEFER`
8. Set the signal processor mask to the appropriate value before invoking the signal handler. This mask is formed by taking the union of the current signal mask (`PPSDCATCHERMASK`), the value of `Sa_mask` specified on the `sigaction` call for the signal being delivered (`PPSDSAMASK`), and then including the signal being delivered (unless the `SA_NODEFER` flag is set). The signal processor mask is set by calling the `sigprocmask` service (`BPX1SPM, BPX4SPM`). Recursive calls to the SIR can occur after calling the `sigprocmask` service here to unblock signals. Therefore, the SIR must use the local copy of the `BPXYPPSD` macro after calling the `sigprocmask` service.
9. Conform to the language-dependent requirements for invoking signal-handlers.
10. On return from the signal handler, call the `sigprocmask` service to set the signal processor mask to the interrupted value that was saved in the `BPXYPPSD` field `PPSDCURRENTMASK` on entry to this SIR.
11. Use the `CSRL16J` MVS service to load 16 registers and jump to the address that was interrupted by the signal.

The use of the `Default_terminate_signal_set` is to indicate to the kernel which signals intercepted by the SIR cause the process to end. For example, a user might wish to intercept the `SIGUSR1` signal, but rather than performing the default of termination, the user might wish to have a message issued and then the signal thrown away (ignored). In this case, the user would turn the corresponding bit on in the `Default_override_signal_set` and off in the `Default_terminate_signal_set`. This bit set combination tells the kernel not to interrupt functions that return an `EINTR`.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm”](#) on page 30

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#)
- [“pthread\\_quiesce \(BPX1PTQ, BPX4PTQ\) — Quiesce threads in a process” on page 598](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered” on page 851](#)

**Characteristics and restrictions**

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413.](#)

**Examples**

See [“BPX1MSS \(mvssigsetup\) example” on page 1266.](#)

## MVSThreadAffinity (BPX1TAF, BPX4TAF) — MVS thread affinity service

---

**Function**

The MVSThreadAffinity callable service allows a task created with pthread\_create to request that a user-defined assembler routine is to be run on a specified target pthread. The requesting and target pthread must have been created with pthread\_create, and both threads must be under the same initial pthread-creating thread (IPT). The requesting pthread is blocked until the requested routine has been run. The target pthread may be the IPT.

This service provides the ability for a program to manage MVS resources under the target pthread or IPT, instead of under the requesting pthread. Resources that can be managed with this service include load modules, opened data sets, and other MVS resources with task affinity.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TAF):</b>	31-bit
<b>AMODE (BPX4TAF):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1TAF,(Routine_address,
              Parameter_list,
              Thread_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TAF with the same parameters. Routine\_address and Parameter\_list are 64-bit pointer fields.

## Parameters

### Routine\_address

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the routine to which control is to be passed on the target pthread. The requesting pthread is responsible for ensuring that the routine to be run is in memory when it is called, and remains there until the call is complete.

### Parameter\_list

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the routine parameter list. The value in this fullword is passed in register 1 when the specified routine receives control. If the routine does not require parameters, specify 0.

### Thread\_ID

Supplied parameter

#### Type:

Character string

#### Character set:

N/A

#### Length:

8 bytes

The name of an 8-byte field that contains the target pthread under which the routine is to run. This is the value returned by the pthread\_self service, or pointed to by the PTXL field PTXLTHIDPTR provided by the pthread\_exit\_and\_get service. (See [“BPXYPTXL — Map the parameter list for pthread\\_create”](#) on page 1141.) A value of all zeros will target the IPT.

### Return\_value

Returned parameter

#### Type:

Integer

#### Character set:

N/A

#### Length:

Fullword

The name of a fullword in which the MVSThreadAffinity service returns 0 if the request completes successfully, or -1 if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Character set:**

N/A

**Length:**

Fullword

The name of a fullword in which the MVSThreadAffinity service stores the return code. The MVSThreadAffinity service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errnos) in *z/OS UNIX System Services Messages and Codes*. The MVSThreadAffinity service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	A bad address was received as an argument of the call; or the specified routine experienced an abnormal end or program check that was not handled by the routine's recovery. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRBadAddress and JRRoutineError.
<b>EAGAIN</b>	Another pthread within the process has this call pending for the specified pthread. At most one pthread can request this service at a time for a given pthread. The requesting pthread can try again later when the current pending call is complete.
<b>EACESS</b>	A task other than a pthread-created task or IPT is not permitted to perform this service.
<b>EINVAL</b>	A thread with the specified thread ID was not found. The reason code accompanying this return code is JRThreadNotFound.
<b>EMVSERR</b>	The passed Routine_address is zero or greater than X'7FFFFFFF'. The reason code accompanying this return code is JrInvalidRoutine.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Character set:**

N/A

**Length:**

Fullword

The name of a fullword in which the MVSThreadAffinity service stores the reason code. The MVSThreadAffinity service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see Reason codes in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The specified routine to be executed receives control with the following attributes:
  - Problem program state
  - Key of the target pthread task
  - AMODE 31
  - Primary ASC mode

2. The register usage on entry to the specified routine is:

- R0: Undefined
- R1: Address of Parameter\_list, as specified by the caller of the MVSThreadAffinity service
- R2–R12: Undefined
- R13: Address of a 72-byte work area with which the routine gains control. For AMODE 64 callers, the work area is 144 bytes long.
- R14: The return address from the specified routine to the MVSThreadAffinity. This address must be preserved by the invoked routine.
- R15: Address of the invoked routine

3. Only tasks created with pthread\_create or the IPT can invoke this service. If a task that is not an IPT or a pthread-created task requests this service, it receives an EACCES return code.

## Related services

- [“mvsiptaffinity \(BPX1IPT, BPX4IPT\) — Run a program on the IPT thread ” on page 476](#)
- [“pthread\\_exit\\_and\\_get \(BPX1PTX, BPX4PTX\) — Exit and get a new thread ” on page 588](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1TAF \(MVSThreadAffinity\) example” on page 1301](#).

# mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup

## Function

The mvsunsigsetup callable service deletes the task's signal set up established by the mvssigsetup service (see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)). The parameters specified in the mvssigsetup service are returned by the mvsunsigsetup service. The signal actions for all signals in the process set by the sigaction service (see [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)) are set to default action SIG\_DFL.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MSD):</b>	31-bit
<b>AMODE (BPX4MSD):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MSD,(Signal_interface_routine_address,
              User_data,
              Default_override_signal_set,
              Default_terminate_signal_set,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MSD with the same parameters. The `Signal_interface_routine_address` parameter is a doubleword.

## Parameters

### **Signal\_interface\_routine\_address**

Returned parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) return area where `Signal_interface_routine_address`, set by the `mvssigsetup` service, is returned.

### **User\_data**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword return area where `User_data`, set by the `mvssigsetup` service, is returned.

### **Default\_override\_signal\_set**

Returned parameter

**Type:**

Character string

**Character set:**

No restriction

**Length**

8 bytes

The name of an 8-byte area where `Default_override_signal_set`, set by the `mvssigsetup` service, is returned.

### **Default\_terminate\_signal\_set**

Returned parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

8 bytes

The name of an 8-byte area where `Default_terminate_signal_set`, set by the `mvssigsetup` service, is returned.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the mvsunsigsetup service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvsunsigsetup service stores the return code. The mvsunsigsetup service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The mvsunsigsetup service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EMVSINITIAL	The service failed (JRNotSigSetup).

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the mvsunsigsetup service stores the reason code. The mvsunsigsetup service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)

**Characteristics and restrictions**

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413](#).

**Examples**

See [“BPX1MSD \(mvsunsigsetup\) example” on page 1265](#).

## **nice (BPX1NIC, BPX4NIC) — Change the nice value of a process**

---

**Function**

The nice callable service changes the nice value of the calling process.

## Requirements

### Operation

#### Authorization:

### Environment

Supervisor or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1NIC):

31-bit

#### AMODE (BPX4NIC):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1NIC,(Nice_change,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4NIC with the same parameters.

## Parameters

### Nice\_change

Supplied parameter

#### Type:

Signed Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates the relative change in the nice value of the calling process.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the nice service returns -1 if it is not successful. If it is successful, the nice service returns the new nice value minus NICE\_ZERO. The constant NICE\_ZERO is defined in the BPXYCONS macro (see [“BPXYCONS — Constants used by services”](#) on page 1062).

Because the nice service can return the value -1 on successful completion, it is necessary to set the Return\_code parameter to 0 before a call to nice. If nice returns the value -1, the Return\_code parameter can be checked to see if an error occurred or if the service was successful.

### Return\_code

Returned parameter



**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the nice service stores the return code. The nice service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The nice service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EPERM</b>	The nice_change value is negative, and the calling process does not have the appropriate privileges (see <a href="#">“Authorization”</a> on page 8).
<b>EMVSSAF2ERR</b>	A security product internal error has occurred. Consult Reason_code for the exact reason for the error.
<b>ENOSYS</b>	The system does not support this function. Your installation has chosen not to enable it.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the nice service stores the reason code. The nice service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. A process's nice value is a non-negative integer for which a more positive value would result in a lower CPU priority. A minimum nice value of 0 and a maximum value of (2\*NICE\_ZERO)-1 are imposed on all processes. If the specified nice\_change value would result in a nice value that is outside this range, the nice value is set to the limit value. The default nice value for all processes is set to the constant value NICE\_ZERO, which is defined in BPXYCONS.
2. If the specified nice\_change value is negative, the value would result in a lowering of a process's nice value, thus giving the process a higher CPU priority. Only processes with the appropriate privileges (see [“Authorization”](#) on page 8) can lower their nice values.
3. The changing of a process's nice value has the same effect on a process's priority value, because they both represent the process's relative CPU priority. For example, increasing the nice value of a process to its maximum value of (2\*NICE\_ZERO) - 1 has the effect of setting its priority value via the setpriority service to its maximum value (19), and will be reflected on the nice, getpriority, and setpriority services.
4. If the calling process is in a multiple-process address space, each of the processes in the address space has its nice value changed by the call to the nice service.
5. If the ENOSYS return code is received, your installation does not support this service. Contact your system administrator if you require activation of this service.
6. To set up the nice service, see [Enabling nice\(\)](#), [setpriority\(\)](#), and [chpriority\(\)](#) support in *z/OS UNIX System Services Planning*.

**Related services**

- [“setpriority \(BPX1SPY, BPX4SPY\) — Set the scheduling priority of a process”](#) on page 779

- [“getpriority \(BPX1GPY, BPX4GPY\) — Get the scheduling priority of a process ” on page 300](#)

## Characteristics and restrictions

If the calling process is running in a multiple-process address space, the nice values of all the processes in the address space are changed upon successful completion of the nice service.

## Examples

See [“BPX1NIC \(nice\) example” on page 1266](#).

# oe\_env\_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute

---

## Function

The oe\_env\_np service examines, changes, or examines and changes an environmental attribute. The environmental attribute to be processed is determined by the value that is specified by the Function\_code parameter.

## Requirements

### Operation

#### Authorization:

#### Dispatchable unit mode:

#### Cross memory mode:

#### AMODE (BPX1ENV):

#### AMODE (BPX4ENV):

#### ASC mode:

#### Interrupt status:

#### Locks:

#### Control parameters:

### Environment

Supervisor state or problem state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1ENV, (Function_code,
               InArgCount,
               InArgListPtr,
               OutArgCount,
               OutArgListPtr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ENV with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Function\_code

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword specifying a numeric value that identifies the environmental attribute that the caller wants to examine, change, or both examine and change. Each environmental attribute has a specific Function\_code value; these attributes are defined in the BPXYCONS macro. See [“BPXYCONS — Constants used by services”](#) on page 1062.

Constant	Function
<b>DFP_CLEANUP_EXIT_REG</b>	Registers a DFP cleanup exit that is to be called during process cleanup processing. No other input parameters are applicable for this function. Upon success, a return value of zero is supplied. No unique error codes apply to this function code.
<b>ENQWAIT_PROCESS</b>	Determines the kernel behavior when pthread_quiesce (freeze or term) and pthread_cancel encounter threads in MVS ENQ waits.
<b>FREEZE_EXIT_REG</b>	Registers or deregisters a user exit that is to be given control when a pthread_quiesce(freeze_exit) call is made.
<b>MVS_USERID</b>	Retrieves the MVS user ID of the invoker.
<b>ENV_TOGGLE_SEC</b>	Toggles the task-level security.
<b>ENV_STOR_SERVICE</b>	Modifies storage attributes of an address space.
<b>SHUTDOWN_REG</b>	Registers the caller for special treatment at OMVS shutdown time.
<b>WRITE_DOWN</b>	Sets, resets, or queries the setting of the write-down privilege in the target ACEE.
<b>PIDXFER_QUERY</b>	Determines whether the current process image was the result of a PIDXFER-type exec.
<b>QUERY_MODE</b>	Returns the AMODE, RMODE, and AMODE capability of a target PID.
<b>MUST_STAY_CLEAN</b>	Sets or queries the address space MUST_STAY_CLEAN state. Once this state is set, any loads or execs are prevented to files that reside in uncontrolled libraries.

The value that is specified for the Function\_code also determines the number and length of input and output parameters. See the usage notes for details on defined function codes.

**InArgCount**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword specifying a numeric value that indicates the number of parameters pointed to by the InArgListPtr parameter. If no input arguments are required, specify the name of a fullword that contains 0. If 0 is specified, no environmental attributes are changed.

**InArgListPtr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) containing an address that points to an array of addresses that point to parameters. If the value that is specified for InArgCount is 0, the value that is specified for the InArgListPtr is ignored. See the usage notes for details on how to specify input parameters.

**OutArgCount**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword specifying a numeric value that indicates the number of parameters pointed to by the OutArgListPtr parameter. If no output arguments are required, specify the name of a fullword that contains 0. If 0 is specified, no environmental attributes are examined.

**OutArgListPtr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) containing an address that points to an array of addresses that point to parameters. If the value that is specified for OutArgCount is 0, the value that is specified for the OutArgListPtr is ignored. See the usage notes for details on how to specify input parameters.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the oe\_env\_np service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the oe\_env\_np service stores the return code. The oe\_env\_np service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The oe\_env\_np service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EFAULT**

The InArgListPtr, OutArgListPtr or associated parameter lists point to a location that is partially or completely outside of the addressable storage range.

Return_code	Explanation
<b>EINVAL</b>	The Function_code value that is specified is not defined, or the InArgCount or OutArgCount parameter contains an incorrect count for the specified Function_code. The following reason codes can accompany the return code: JRFuncUndefined, JRBadArgCount or JRBadInputValue. If the SHUTDOWN_REG function was requested, the following reason codes can accompany the return code: JRJSTMustBeRegistered, JRAreadyInShutDown, JRBlockPermAlreadyRegistered, JRBlockPermNotRegistered, JRBadInputValue, or JRAreadyInShutDown.
<b>EMVSSAF2ERR</b>	There was an internal error in the security product. The hexadecimal reason code value contains the 2-byte security product return code xx and reason code yy.
<b>EMVSERR</b>	The BPX.DAEMON FACILITY class profile is not defined. Reason code JRNoDaemon can accompany this return code.
<b>EPERM</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• The calling process does not have the appropriate privilege to perform the requested operation. The reason code JROK can accompany this return code. If the SHUTDOWN_REG function was requested, the caller must be given read permission to the BPX.SHUTDOWN resource in the FACILITY class.</li> <li>• A call was made to register as a permanent process or job but the calling process was started with the respawn attribute. Reason code JRRespawnNotAllowed can accompany this return code.</li> <li>• A call was made to register a notify exit from an unauthorized caller that is running in a TSO address space. This function is not allowed.</li> </ul>
<b>ENOSYS</b>	The implementation does not support this memory locking interface. Reason code JRNotBpxStorSwap can accompany this return code.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

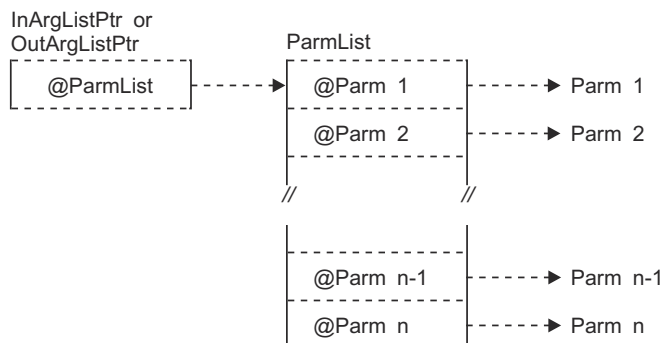
The name of a fullword in which the oe\_env\_np service stores the reason code. The oe\_env\_np service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the value that is specified by InArgCount or OutArgCount is 0, the corresponding InArgListPtr or OutArgListPtr is ignored. They must still be specified, but the value that is contained in the named field is irrelevant.
2. The oe\_env\_np service can examine, change, or examine and change environmental attributes based on the argument counts that are specified by the caller. If only an InArgCount is specified, an environmental attribute is changed, but the previous value is not returned. If only an OutArgCount is specified, the current setting of an environmental attribute is examined and returned but not changed. If both an InArgCount and OutArgCount are specified, the environmental attribute is changed and the previous setting is returned. If neither InArgCount nor OutArgCount are specified, no environmental attributes are examined or changed (NOOP), and the oe\_env\_np service sets Return\_value to 0.

3. The InArgListPtr and OutArgListPtr parameters each contain a fullword address that points to an array. The argument count (InArgCount and OutArgCount) defines the number of elements in each of these arrays. Each element in the arrays contains a fullword address that points to a parameter. The length of each parameter varies according to the Function\_code specified.

The following figure is an example of an input or output parameter list as specified by the InArgListPtr and OutArgListPtr parameter.



4. Table 12 on page 504 defines the number of input and output arguments (if specified) and the scope of each defined Function\_code.

The scope of an environmental attribute is the range of influence the attribute has in the kernel. The highest level of scope is ADDRESS SPACE: these attributes have influence over all processes and threads in an address space. The next level of scope is PROCESS: these attributes have influence over a single process. The lowest level of scope is THREAD: these attributes have influence over a single thread.

Table 12. Number of input and output arguments and the scope of each defined Function_code			
Function_code	# Input Args	# Output Args	Scope
ENQWAIT_PROCESS	1	1	Process
FREEZE_EXIT_REG	1	1	Process
MVS_USERID	0	1	Thread
ENV_TOGGLE_SEC	0	0	Thread
ENV_STOR_SERVICE	1	0	Address space
SHUTDOWN_REG	4	0	Process or address space
WRITE_DOWN	1	0 or 1	Address space or thread
PIDXFER_QUERY	0	1	Process
QUERY_MODE	1	3	Address space (AMODE, RMODE) or process (AMODE capability)
MUST_STAY_CLEAN	1	1	Address space

5. Function\_code and argument definitions:

- ENQWAIT\_PROCESS

The purpose of the ENQWAIT\_PROCESS Function\_code is to register with the kernel the behavior desired when a pthread\_quiesce(freeze or term) or pthread\_cancel encounters a thread in an ENQ wait in the caller's process.

When ENQWAIT\_PROCESS is disabled, the kernel does not interrupt threads that are found in ENQ waits. This means that pthread\_quiesce(freeze or term) and pthread\_cancel events are not delivered to a thread until after the ENQ wait has completed. This is the default behavior for all processes.

When ENQWAIT\_PROCESS is enabled, the kernel interrupts threads that are found in ENQ waits. The kernel delivers pthread\_quiesce(freeze) events to threads by scheduling an IRB on top of the SVRB for the ENQ wait, and freezing the thread from the IRB. For pthread\_quiesce(term) or pthread\_cancel events, the kernel abends threads in ENQ waits with a retryable 422 abend, reason code X'00000189'.

To get pending pthread\_cancel and pthread\_quiesce(term) events delivered, applications that invoke ENQ need to do the following:

- Establish an ESTAE before invoking ENQ.
- Have the ESTAE retry and check for the 422 abend with reason code X'00000189'.
- When the abend is detected, call sigprocmask and block all signals. On return from sigprocmask the pthread\_cancel or pthread\_quiesce(term) events are delivered.

When a pthread\_cancel interrupts a thread in an ENQ, the target thread is abended (S0422-189). If an ESTAE has not been established or just percolates, the entire process is terminated. This behavior is required for standards compliance.

- Input arguments:
  - First argument (fullword):
    - 0**

Disable ENQ wait interrupt support in the caller's process.
    - 1**

Enable ENQ wait interrupt support in the caller's process.
- Output arguments:
  - First argument (fullword):
    - 0**

ENQ wait interrupt support is disabled in the caller's process.
    - 1**

ENQ wait interrupt support is enabled in the caller's process.

#### • FREEZE\_EXIT\_REG

The purpose of the FREEZE\_EXIT\_REG Function\_code is to register with the kernel the address of an exit that is to get control when the Freeze-Exit function code of the pthread\_quiesce service is requested. The exit gets control once on each thread processed by the pthread\_quiesce(Freeze\_Exit) service.

If a FREEZE\_EXIT\_REG is registered, the kernel gives control to the specified exit as a result of the pthread\_quiesce(freeze\_exit) call.

The user exit is given control while the pthread\_quiesce service is still in progress. The user exit should not attempt to use any service that alters or terminates the current process. No callable services should be requested. If such services are attempted, the results are unpredictable.

The register usage on entry to the user exit is:

- R0: Undefined
- R1: Address of the parameter list defined by PpsdSIRParms. The first word of this parameter list is the address of the Ppsd.
- R2–R15: Undefined

When the exit returns, there are no expected return values or codes. The exit routine should terminate via SVC 3.

- Input arguments:

- First argument (fullword):

**address**

Pthread\_quiesce(Freeze\_Exit) exit address.

Clear the pthread\_quiesce(Freeze\_exit) address. Specifying zero unregisters an exit address.

- Output arguments:

- First argument (fullword):

**address**

Pthread\_quiesce(Freeze\_Exit) exit address.

No exit was registered with the kernel.

- **MVS\_USERID**

The purpose of the MVS\_USERID Function\_code is to query the current MVS identity of the caller. The MVS user ID that is returned can be affected by the presence of a task-level security environment. If a task-level security environment has been created by the pthread\_security\_np service, the user ID that is associated with the task is returned.

The InArgCount must be 0, and the OutArgCount must be 1.

- Input arguments:

- First argument (None)

- Output arguments:

- First argument (Char 8)

**The current MVS user ID**

8-character user ID padded on the right with blanks.

- **ENV\_TOGGLE\_SEC**

The purpose of the ENV\_TOGGLE\_SEC Function\_code is to toggle the task-level security. If the calling task has a task-level security environment, it is saved, and the task security is set back to the process level. If the calling task has a saved security environment and currently has no task-level security, the saved security environment is reinstated. If the calling task has not made a prior call to pthread\_security\_np, this call has no effect.

There are no additional input or output arguments, so the InArgCount and the OutArgCount must be 0.

- **ENV\_STOR\_SERVICE**

The purpose of the ENV\_STOR\_SERVICE Function\_code is to modify the storage attributes of the caller's address space. The caller's address space cannot be made swappable unless it has previously been made non-swappable by this function. If the function is called to make an address space non-swappable and the current address space has already been made non-swappable by this function, the call is ignored and the address space remains non-swappable.

The InArgCount must be 1, and the OutArgCount must be 0.

- Input arguments:

- First argument (Structure):

ENV\_STOR\_FLAGS (Supplied Parameter)

ENV\_STOR\_FLAGS can be set to the following values defined in BPXYCONS macro:

**Value**

**Definition**

**BPX\_SWAP**

Makes the address space swappable. The caller needs at least READ access to the BPX.STOR.SWAP resource in the FACILITY class.



Value	Definition
<b>BPX_NONSWAP</b>	<p>Makes the address space non-swappable. The caller needs at least READ access to the BPX.STOR.SWAP resource in the FACILITY class.</p> <p>When an application makes an address space non-swappable, it may cause additional real storage in the system to be converted to preferred storage. Preferred storage cannot be configured offline. Use of this service can therefore reduce an installation's ability to reconfigure storage in the future. Any application using this service should warn the customer about this side effect in the installation documentation.</p>

- Output arguments:
  - First argument (None)

- SHUTDOWN\_REG

The purpose of the SHUTDOWN\_REG Function\_code is to request special treatment for the caller at OMVS shutdown time.

The SHUTDOWN\_REG exit receives control in the caller's address space with it being both the home and primary address space. For a process-level registration, the exit runs in task mode from an IRB running on the initial thread task of the process that called the BPXnENV registration service. For a job-level registration, it runs on the initial thread task of the first process in the address space. The authorization state of the exit will be the same PSW key and PSW state of the caller of BPXnENV. The exit receives no parameters. Because the exit is driven from an IRB, it might not be able to issue other z/OS UNIX callable services; therefore, it should not depend on doing so.

The following is the register usage on entry to the exit:

- R0–R12: Undefined
- R13: Address of a 96-byte work area in the same key as the caller that registered the exit
- R14: The return address from the exit to the operating system. The exit must preserve this address to be used to return to the operating system.
- R15: The address of the exit

There are constants defined in BPXYCONS for use with the SHUTDOWN\_REG function. See [“BPXYCONS — Constants used by services” on page 1062](#).

The InArgCount must be 4, and the OutArgCount must be 0.

- Input arguments:
  - First parameter (fullword):

The first parameter contains a value that indicates the type of shutdown registration being requested:

Value	Definition
<b>1</b>	Register as a blocking process or job.
<b>2</b>	Register as a permanent process or job.
<b>3</b>	Unregister as a blocking process or job.
<b>4</b>	Unregister as a permanent process or job.
<b>5</b>	Register for notification of shutdown.
<b>6</b>	Unregister for notification of shutdown.

- Second parameter (fullword):

The second parameter contains a value that indicates the scope of shutdown registration being requested:

Value	Definition
<b>1</b>	Register for the calling job.
<b>2</b>	Register for the calling process.

The values in the first two parameters are mutually exclusive; they cannot be combined.

- Third parameter (fullword):

The third parameter contains a value that indicates the registration options being requested:

Value	Definition
<b>1</b>	Block system calls that are waiting for restart (valid for permanent registration only).
<b>2</b>	Abend system calls during shutdown/restart (valid for permanent registration only). This option is mutually exclusive with option value 1.
<b>4</b>	Send a SIGTERM signal when shutdown is initiated.
<b>8</b>	Invoke a specified exit when shutdown is initiated. This option is mutually exclusive with option 4 (send a SIGTERM).

- Fourth parameter (fullword):

The fourth parameter contains the address of the exit that is to receive control at shutdown time, or 0. On the 64-bit interface, the exit address is an 8-byte field on the registration call. However, the exit address is still a 31-bit address.

- Output arguments:

- (None)

- **WRITE\_DOWN**

The purpose of the WRITE\_DOWN Function\_code is to set, reset, or query the setting of the write-down privilege in the target ACEE.

The InArgCount must be 2, and the OutArgCount must be 0 or 1.

- Input arguments:

- First argument (Fullword):

The first argument contains a value that indicates the type of operation to be performed:

Value	Definition
<b>0</b>	Query the current setting of the write-down privilege.
<b>1</b>	Activate the write-down privilege.
<b>2</b>	Deactivate the write-down privilege.
<b>3</b>	Reset the write-down privilege to its default value.

- Second argument (fullword):

The second argument contains a value that indicates the scope of the write-down privilege operation to be performed:

Value	Definition
<b>1</b>	Perform write-down operation on address-space level ACEE (WD_SCOPE_AS).
<b>2</b>	Perform write-down operation on task-level ACEE (WD_SCOPE_THD).

- Output arguments:

- First argument (fullword):

This argument is only returned when the OutArgCount is 1. If the OutArgCount is 0, this argument is ignored.

Value	Definition
<b>0</b>	Write-down privilege is inactive for ACEE (WD_IS_INACTIVE).
<b>1</b>	Write-down privilege is active for ACEE (WD_IS_ACTIVE).

- PIDXFER\_QUERY

The purpose of the PIDXFER\_QUERY Function\_code is to determine if the caller's current process image was created as the result of a PIDXFER-type exec.

The InArgCount must be 0, and the OutArgCount must be 1.

- Input arguments:

- First argument: None.

- Output arguments:

- First argument (fullword):

Value	Definition
<b>0</b>	The current process image was not the result of a PIDXFER exec (PIDXFER_NO).
<b>1</b>	The current process image was the result of a PIDXFER exec (PIDXFER_YES).

- QUERY\_MODE

The purpose of the QUERY\_MODE Function\_code is to query:

- The addressing mode (AMODE) of the target PID
- The residency mode (RMODE) of the target PID
- The ability of the address space in which the PID is running to support an AMODE 64 program; in other words, whether storage can be obtained above the bar

The service records the modes at the time the process was created (dubbed), or at the time of the last executed program. If a process has multiple threads, the service reports on the AMODE of the initial thread of the process.

The InArgCount must be 1, and the OutArgCount must be 3.

- Input arguments:

- First argument: PID of the target process.

- Output arguments:

- First argument (fullword):

The first argument contains a value that indicates the AMODE of the target process:

Value	Definition
<b>1</b>	24-bit AMODE
<b>2</b>	31-bit AMODE
<b>3</b>	64-bit AMODE

- Second argument (fullword):

The second argument contains a value that indicates the RMODE of the target process:

Value	Definition
<b>1</b>	24-bit RMODE
<b>2</b>	31-bit RMODE
<b>3</b>	64-bit RMODE

- Third argument (fullword):

The third argument contains a value that indicates the AMODE capability of the target process:

Value	Definition
<b>1</b>	24-bit AMODE capability
<b>2</b>	31-bit AMODE capability
<b>3</b>	64-bit AMODE capability

- **MUST\_STAY\_CLEAN**

The purpose of the MUST\_STAY\_CLEAN Function\_code is to ensure that a process and its children are, and will remain program-controlled. A process must be program-controlled in order to enable the MUST\_STAY\_CLEAN state. If a process is not program-controlled, this service will fail with a return code of EMVSERR and a reason code of JRENVDIRTY. Also, message BPXP015I is written to the console indicating the program that made the process uncontrolled. Once set, a process will not be able to load, execute, or spawn any files that are not from program-controlled libraries. Since this state is also propagated to children during a fork or spawn, the processes involved can fully trust one another.

The typical usage is for a process to set this state, create any children, and then in any child processes query the state. In order to trust the parent the child must issue the query service before any other security-related services are used in the child process. (For a list of these security services see *Setting up the UNIX-related FACILITY and SURROGAT class profiles in z/OS UNIX System Services Planning*.) In this way the child can be sure that it has inherited the state instead of the state being set by an action of the child. If the child recognizes that the state is enabled, all processes involved can trust one another. Usage of this service to enable the MUST\_STAY\_CLEAN state requires the BPX.DAEMON FACILITY class profile to be defined. If it is not defined the service will fail with a return code of EMVSERR and a reason code of JRNODAEMON.

– Input arguments:

- First argument (fullword):

- 0** Query the current MUST\_STAY\_CLEAN state. The current state is returned in the first output argument.
- 1** Enable the MUST\_STAY\_CLEAN state.

– Output arguments:

- First argument (fullword):

- 0** The MUST\_STAY\_CLEAN state is disabled.
- 1** The MUST\_STAY\_CLEAN state is enabled.
- 2** The MUST\_STAY\_CLEAN state is conditional

- If the MUST\_STAY\_CLEAN service returns EMVSSAF2ERR, then the propagated failing 1-byte return code and 1-byte reason code of the IRRENS00 service is found in the last two bytes of the reason code returned by BPX1ENV/BPX4ENV.

7. A returned state of MSC\_ENABLED indicates that the state was set by this service and will continue even after exec ( ) s that cause a job step to terminate. A returned state of MSC\_ENABLED\_COND indicates a previous security service, such as BPX1PWD, enabled the clean state and will be reset when the next job step runs.

## Related services

- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#)
- [“pthread\\_quiesce \(BPX1PTQ, BPX4PTQ\) — Quiesce threads in a process” on page 598](#)
- [“pthread\\_security\\_np, pthread\\_security\\_applid\\_np \(BPX1TLS, BPX4TLS\) — Create or delete thread-level security” on page 601](#)

## Characteristics and restrictions

1. Users of the blocking and permanent registration options of the SHUTDOWN\_REG function must meet one of the following requirements:
  - The calling address space must be a system started task address space.
  - The caller must be running authorized (APF-authorized, system key 0–7, or supervisor state).
  - The caller must be a privileged z/OS UNIX process. It must have either superuser identity or read permission to the BPX.SHUTDOWN profile in the FACILITY class.
2. For the write-down privilege to be activated, the user ID in the target ACEE must be permitted to the IRR.WRITEDOWN.BYUSER profile in the FACILITY class. The FACILITY class must be active and RACLISTed, and the SETROPTS MLS option must be active.

## Examples

See [“BPX1ENV \(oe\\_env\\_np\) example” on page 1233](#).

# open (BPX1OPN, BPX4OPN) — Open a file

---

## Function

The open callable service gains access to a file and creates a file descriptor for it.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1OPN)</b>	31-bit
<b>AMODE (BPX4OPN)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1OPN, (Pathname_length,  
               Pathname,  
               Options,  
               Mode,  
               Return_value,  
               Return_code,  
               Reason_code)
```

AMODE 64 callers use BPX4OPN with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the path name of the file.

### Pathname

Supplied parameter

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Pathname\_length parameter

The name of a field that contains the name of the file to be opened. The length of this field is specified in Pathname\_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

### Options

Supplied parameter

#### Type

Structure

#### Length

Fullword

The name of a fullword that contains the binary flags that describe how the file is to be opened. The flags are as follows:

#### O\_APPEND

Positions the file offset at the end of the file before each write operation.

#### O\_CLOEXEC

Enables the close-on-exec flag for the new file descriptor. Specifying this flag permits a program to avoid additional fcntl F\_SETFD operations to set the FD\_CLOEXEC flag.

#### O\_CREAT

Indicates that the call to the open service has a *mode* argument.

- If the file being opened already exists, O\_CREAT has no effect except when O\_EXCL is also specified. See “O\_EXCL” on page 513.
- If the file being opened does not exist, it is created. The user ID is set to the effective ID of the process and its group ID is set to the group ID of its directory. File permission bits are set according to *mode*.
- If O\_CREAT is specified and the file did not previously exist, a successful open sets the access time, change time, and modification time for the file. It also updates the change time and modification time fields in the parent directory.

When a file is created, the file permission bits as specified in the Mode parameter are modified by the process's file creation mask and then used to set the file permission bits of the file that is being created. For more information about the file creation mask, see “umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask” on page 960.

The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:

- If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
- If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.

### **O\_DIRECT**

Tries to minimize the cache effects of the I/O to and from this file.

### **O\_DIRECTORY**

The open fails if *pathname* is not a directory.

### **O\_EXCL**

If both O\_EXCL and O\_CREAT are specified, the open fails if the file already exists. If both O\_EXCL and O\_CREAT are specified and *pathname* names a symbolic link, the open fails regardless of the contents of the symbolic link.

The check for the existence of the file and the creation of the file if it does not exist is atomic with respect to other threads executing open naming the same file name in the same directory with both O\_EXCL and O\_CREAT set.

If O\_EXCL is set but O\_CREAT is not set, O\_EXCL is ignored.

When a file is created with O\_EXCL, the file permission bits as specified in the Mode parameter are modified by the process's file creation mask. They are then used to set the file permission bits of the file that is being created. For more information about the file creation mask, see “umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask” on page 960.

The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:

- If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
- If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.

### **O\_NOCTTY**

If *pathname* specifies a terminal, the open does not make the terminal the controlling terminal of the process (and the session). If O\_NOCTTY is not specified, the terminal becomes the controlling terminal if the following conditions are true:

- The process is a session leader.
- There is no controlling terminal for the session.
- The terminal is not already a controlling terminal for another session.

**O\_NOFOLLOW**

If the trailing component (for example, the base name) of *pathname* is a symbolic link, the open fails with the error ELOOP.

**O\_NONBLOCK**

Has different meanings, depending on the situation.

- When you are opening a FIFO special file with O\_RDONLY or O\_WRONLY.

If O\_NONBLOCK is specified, read-only open returns immediately. A write-only open returns with an error if no other process has the FIFO open for reading.

If O\_NONBLOCK is not specified, a read-only open blocks until another process opens the FIFO for writing. A write-only open blocks until another process opens the FIFO for reading.

- When you are opening a character special file that supports a nonblocking open, O\_NONBLOCK controls whether subsequent reads and writes can block.

A *FIFO special file* is a shared file from which the first data written is the first data read.

O\_NONBLOCK is a way of coordinating write and read requests between processes that share a FIFO special file. If no other conditions interfere with opening the file successfully, it works as follows:

- If a file is opened read-only and O\_NONBLOCK is specified, the open request succeeds. Control returns to the caller immediately.
- If a file is opened write-only and O\_NONBLOCK is specified, the open request completes successfully, provided that another process has the file open for reading. If another process does not have the file open for reading, the request ends with Return\_value set to -1.
- If a file is opened read-only and O\_NONBLOCK is omitted, the request is blocked (control is not returned to the caller) until another process opens the file for writing.
- If a file is opened write-only and O\_NONBLOCK is omitted, the request is blocked (control is not returned to the caller) until another process opens the file for reading.

**O\_PATH**

Obtains a file descriptor either to indicate a location in the file system tree or to perform operations that act at the file descriptor level.

**O\_TRUNC**

If the file is successfully opened with O\_RDWR or O\_WRONLY, the file is truncated to zero length if the file exists and is a regular file. The mode and owner of the file are unchanged. Do not use this option with O\_RDONLY. O\_TRUNC has no effect on FIFO special files or directories.

If O\_TRUNC is specified and the file previously existed, a successful open updates the change time and modification time for the file.

Using O\_TRUNC opens the file as though it had been created earlier, but never written into. The mode and owner of the file do not change, except for the change time and modification time. However, the contents of the file are discarded. The file offset, which indicates where the next write is to occur, points to the first byte of the file.

**O\_SYNC**

Forces synchronous update. If this flag is 1, every write operation on the file is written to permanent storage. That is, the file system buffers are forced to permanent storage. Also see [fsync](#).

The program is assured that all data for the file was written to permanent storage on return from a function, which performs a synchronous update.

When O\_SYNC is set, the program is assured that all data updates have been written to permanent storage.

The options are mapped by the BPXYOPNF macro; see [“BPXYOPNF — Map flag values for open”](#) on page 1113.



**Mode**

Supplied parameter

**Type**

Structure

**Length**

Fullword

The name of a fullword in which the mode field is specified. The mode field, which is mapped by BPXYMODE, specifies the file type and the permissions the caller grants to itself, to its groups, and to any user. See [“BPXYMODE — Map the mode constants”](#) on page 1106.

If O\_CREAT or O\_EXCL is not specified on the Options parameter, the Mode parameter is ignored.

The file type is mapped by the BPXYFTYP macro; see [“BPXYFTYP — File type definitions”](#) on page 1078.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the open service stores the file descriptor if the file was opened successfully, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the open service stores the return code. The open service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The open service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Reasons for being denied access include these: <ul style="list-style-type: none"> <li>• The calling process does not have permission to search one of the directories specified in the Pathname parameter.</li> <li>• The calling process does not have permission to open the file in the way specified by the Options parameter.</li> <li>• The file does not exist. Write permission for the parent directory in which the file would have been created is denied to the calling process.</li> <li>• The truncate option was specified but the process does not have write permission for the file.</li> </ul>
<b>EAGAIN</b>	Resources were temporarily unavailable.
<b>EBUSY</b>	The path name specifies a manager pseudoterminal that is either already in use or for which the corresponding subsidiary is open, or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation

Return_code	Explanation
<b>EEXIST</b>	The exclusive create option (O_EXCL) was specified, but the file already exists. The following reason code can accompany the return code: JRFileExistsExclFlagSet.
<b>EFBIG</b>	A request to create a new file is prohibited because the file size limit for the process is set to 0.
<b>EINTR</b>	The open operation was interrupted by a signal.
<b>EINVAL</b>	The Options parameter does not specify a valid combination of the O_RDONLY, O_WRONLY and O_TRUNC bits; or the file type that was specified in the Mode parameter is not valid. The following reason codes can accompany the return code: JRInvOpenFlags and JROpenFlagConflict.
<b>EISDIR</b>	The file that is specified by Pathname is a directory, and the Options parameter specifies write or read/write access. The following reason code can accompany the return code: JRDirWriteRequest.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMFILE</b>	The process reached the maximum number of file descriptors that it can have open or the current pipe limit was exceeded. Refer to the reason code that was provided.
<b>ENAMETOOLONG</b>	The path name is longer than 1023 characters, or a component of the path name is longer than 255 characters. File names cannot be truncated.
<b>ENFILE</b>	The maximum number of file descriptors that can be open was reached.
<b>ENODEV</b>	<p>Typical causes:</p> <ul style="list-style-type: none"> <li>• An attempt was made to open a character special file for a device that is not supported.</li> <li>• An attempt was made to open a character special file for a device that is not yet initialized.</li> </ul> <p>The following reason code can accompany the return code: JRNoCTTY.</p>
<b>ENOENT</b>	<p>Typical causes:</p> <ul style="list-style-type: none"> <li>• The request did not specify that the file was to be created, but the file named by Pathname was not found.</li> <li>• The request asked for the file to be created, but some component of Pathname was not found, or the Pathname parameter was blank.</li> </ul> <p>The following reason codes can accompany the return code: JREndingSlashOCreat, JRNoFileNoCreatFlag, and JRQuiescing.</p>
<b>ENOSPC</b>	The directory or file system intended to hold a new file has insufficient space.
<b>ENOTDIR</b>	A component of Pathname is not a directory.
<b>ENXIO</b>	The open request specified write-only and nonblock for a FIFO special file, but no process has the file open for reading. For pseudoterminals, it can mean that the minor number that is associated with the path name is too large.

Return_code	Explanation
<b>EPERM</b>	The caller is not permitted to open the specified subsidiary pseudoterminal; or the corresponding manager is not yet open. EPERM is also returned if the subsidiary is closed with HUPCL set, and an attempt is made to reopen it.
<b>EROFS</b>	The Pathname parameter names a file on a read-only file system, but options that would allow the file to be altered were specified: write-only, read/write, truncate, or (for a new file), create. The following reason codes can accompany the return code: JRReadOnlyFileSetWriteReq and JRReadOnlyFileSetCreatReq.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the open service stores the reason code. The open service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. Some remote NFS clients use share reservations to open files on Windows in such a way that no one else can open that file until the first program has finished and closed the file. *Share reservations* prevent others from concurrently opening the same file in certain ways. The share reservations are Deny\_Read, Deny\_Write, Deny\_Both, and Deny\_None. If a file is open by an NFS client and a subsequent attempt to open it for reading or writing is made which is denied by the NFS client's share reservation, the open request either blocks or returns with EBUSY, depending on the specified O\_NonBlock option. Conversely, if a file is open by a local program and an NFS client then attempts to open it and deny the type of access that is already established, the client's open request will fail.

**Related services**

- [“close \(BPX1CLO, BPX4CLO\) — Close a file ” on page 121](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor ” on page 241](#)
- [“fsync \(BPX1FSY, BPX4FSY\) — Write changes to permanent storage” on page 247](#)
- [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset ” on page 406](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)
- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)

**Characteristics and restrictions**

See [“Usage notes” on page 517](#).

**Examples**

See [“BPX1OPN \(open\) example” on page 1267](#).

## MVS-related information

The Execution access requested bit (OPNFEXEC) is used by the exec service to verify that the process has permission to run the specified file. (See [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163.](#))

When the open service succeeds, the specified file is treated as read-only for this case.

## openat/openat2 (BPX10AT, BPX40AT) — Open a file with path name parameters relative to a directory file descriptor

---

### Function

The openat/openat2 service uses the specified directory file descriptor as the starting location for the path search when it opens a file.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX10PN)</b>	31-bit
<b>AMODE (BPX40PN)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX10AT,(Pathname_length, /* in z/OS the pathname length is passed first */
             Pathname,
             dirfd,
             Open_how_length,
             Open_how,
             Return_value,
             Return_code,
             Reason_code)
```

AMODE 64 callers use BPX40AT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

### Parameters

#### Pathname\_length

Supplied parameters.

#### Type

Character string.

#### Length

Integer. The allowed range of lengths is the same as the path name lengths on other file system calls.

**Pathname**

Supplied parameter.

**Type**

Character string.

**Length**

Variable.

Pointer to a string that contains the path name to open.

**dirfd**

Supplied parameter.

**Type**

File descriptor.

**Length**

Fullword.

The name of a fullword that contains a file descriptor for the starting directory to search, or the special value `AT_FDCWD`, which means to use the current working directory as the starting point.

**Open\_how\_length**

Supplied parameters.

**Type**

Integer. The minimum value is 24 and the maximum value is 32.

**Open\_how**

The name of a `Open_How` structure that has at least three doublewords as mapped in macro `BPXYOHOW`.

- The first word is `how.flags`, which corresponds to the flags parameter passed on a call to `open()`.
- The second word is `how.mode`, which corresponds to the mode flags passed on `open()`.
- The third word, `how.resolve`, contains the `RESOLVE` parameters that are needed for `openat2()` calls and is left zero when the call is `openat()`.

Supplied parameter.

**Type**

Structure.

**Length**

Minimum 24 bytes, maximum 32 for compatibility with future Linux extensions.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the `openat2` service returns the file descriptor of the newly opened file, or `-1` if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length:**

Fullword

The name of a fullword in which the `openat2` service stores the return code. The `openat2` service returns `Return_code` only if `Return_value` is `-1`. For a list of return code values, see [Return codes](#)

(errno) in *z/OS UNIX System Services Messages and Codes* The openat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>E2BIG</b>	An extension that this kernel does not support was specified in how.
<b>EAGAIN</b>	how.resolve contains either RESOLVE_IN_ROOT or RESOLVE_BENEATH, and the kernel could not ensure that a "." component did not escape from <i>dirfd</i> . The caller can choose to retry the openat2() call.
<b>RESOLVE_CACHED</b>	RESOLVE_CACHED was set and the open operation cannot be performed by using only cached information. Try again without RESOLVE_CACHED set in how.resolve.
<b>EINVAL</b>	An unknown flag or invalid value was specified in how.
<b>EINVAL</b>	size was smaller than any known version of <i>struct open_how</i> .
<b>ELOOP</b>	how.resolve contains RESOLVE_NO_SYMLINKS, and one of the path components was a symbolic link or magic link.
<b>ELOOP</b>	how.resolve contains RESOLVE_NO_MAGICLINKS, and one of the path components was a magic link.
<b>EXDEV</b>	how.resolve contains either RESOLVE_IN_ROOT or RESOLVE_BENEATH, and an escape from the root during path resolution was detected.
<b>EXDEV</b>	how.resolve contains RESOLVE_NO_XDEV, and a path component crosses a mount point.
<b>EACCES</b>	The following list includes some reasons for being denied access. <ul style="list-style-type: none"> <li>• The calling process does not have permission to search one of the directories specified in the Pathname parameter.</li> <li>• The calling process does not have permission to open the file in the way specified by the Options parameter.</li> <li>• The file does not exist and write permission for the parent directory in which the file would have been created is denied to the calling process.</li> <li>• The truncate option was specified but the process does not have write permission for the file.</li> </ul>
<b>EAGAIN</b>	Resources were temporarily unavailable.
<b>EBUSY</b>	The path name specifies a manager pseudoterminal that is already in use or for which the corresponding subsidiary is open, or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
<b>EEXIST</b>	The exclusive create option was specified, but the file already exists. The following reason code can accompany the return code: JRFileExistsExclFlagSet.
<b>EFBIG</b>	A request to create a new file is prohibited because the file size limit for the process is set to 0.
<b>EINTR</b>	The open operation was interrupted by a signal.
<b>EINVAL</b>	The Options parameter does not specify a valid combination of the O_RDONLY, O_WRONLY and O_TRUNC bits; or the file type that was specified in the Mode parameter is not valid. The following reason codes can accompany the return code: JRInvOpenFlags and JROpenFlagConflict.

Return_code	Explanation
<b>EISDIR</b>	The file that was specified by Pathname is a directory, and the Options parameter specifies write or read/write access. The following reason code can accompany the return code: JRDirWriteRequest.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMFILE</b>	The process reached the maximum number of file descriptors it can have open or the current pipe limit was exceeded. Refer to the reason code that was provided.
<b>ENAMETOOLONG</b>	The path name is longer than 1023 characters, or a component of the path name is longer than 255 characters. File names cannot be truncated.
<b>ENFILE</b>	The maximum number of file descriptors that can be open was reached.
<b>ENODEV</b>	<p>Typical causes:</p> <ul style="list-style-type: none"> <li>• An attempt was made to open a character special file for a device that is not supported.</li> <li>• An attempt was made to open a character special file for a device that is not yet initialized.</li> </ul> <p>The following reason code can accompany the return code: JRNoCTTY.</p>
<b>ENOENT</b>	<p>Typical causes:</p> <ul style="list-style-type: none"> <li>• The request did not specify that the file was to be created, but the file named by Pathname was not found.</li> <li>• The request asked for the file to be created, but some component of Pathname was not found, or the Pathname parameter was blank.</li> </ul> <p>The following reason codes can accompany the return code: JREndingSlashOCreat, JRNoFileNoCreatFlag, and JRQuiescing.</p>
<b>ENOSPC</b>	The directory or file system that is intended to hold a new file has insufficient space.
<b>ENOTDIR</b>	A component of Pathname is not a directory.
<b>ENXIO</b>	The open request that is specified write-only and nonblock for a FIFO special file, but no process has the file open for reading. For pseudoterminals, it can mean that the minor number that is associated with the path name is too large.
<b>EPERM</b>	The caller is not permitted to open the specified subsidiary pseudoterminal or the corresponding manager is not yet open. EPERM is also returned if the subsidiary is closed with HUPCL set, and an attempt is made to reopen it.
<b>EROFS</b>	The Pathname parameter names a file on a read-only file system, but options that would allow the file to be altered were specified: write-only, read/write, truncate, or (for a new file), create. The following reason codes can accompany the return code: JRReadOnlyFileSetWriteReq and JRReadOnlyFileSetCreatReq.
<b>Reason_code</b>	Returned parameter.
<b>Type</b>	Integer

**Length**

Fullword

The name of a fullword in which the openat2 service stores the reason code. The openat2 service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. If *pathname* is an absolute path name, then *dirfd* is ignored (unless *how.resolve* contains RESOLVE\_IN\_ROOT, in which case *pathname* is resolved relative to *dirfd*).
2. *Resolve flags* allow trusted programs to restrict how untrusted paths (or paths inside untrusted directories) are resolved.

**RESOLVE\_BENEATH**

Does not permit the path resolution to succeed if any component of the resolution is not a descendant of the directory that is indicated by *dirfd*. Absolute symbolic links and absolute values of *pathname* are rejected. This flag also disables magic-link resolution. To ensure that magic links are not resolved, the caller should explicitly specify RESOLVE\_NO\_MAGICLINKS.

**RESOLVE\_IN\_ROOT**

Treats the directory referred to by *dirfd* as the root directory while resolving *pathname*. Absolute symbolic links are interpreted relative to *dirfd*. If a prefix component of *pathname* equates to *dirfd*, then an immediately following *..* component likewise equates to *dirfd* (just as *./.* is traditionally equivalent to */*). If *pathname* is an absolute path, it is also interpreted relative to *dirfd*.

The effect of this flag is as though the calling process had used *chroot* to temporarily modify its root directory to the directory referred to by *dirfd*. Unlike *chroot*, which changes the file system root permanently for a process, RESOLVE\_IN\_ROOT allows a program to efficiently restrict path resolution on a per-open basis.

This flag also disables magic-link resolution. To ensure that magic links are not resolved, the caller should explicitly specify RESOLVE\_NO\_MAGICLINKS.

**RESOLVE\_NO\_MAGICLINKS**

Disallows all magic-link resolution during path resolution.

*Magic links* are symbolic link-like objects. Examples of magic links include */proc/[pid]/exe* and */proc/[pid]/fd/\**. For more information, see [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name” on page 900](#).

Unknowingly opening magic links can be risky for some applications. Examples of such risks include the following scenarios:

- If the process that is opening a *pathname* is a controlling process that currently has the no controlling terminal, then opening a magic link inside */proc/[pid]/fd* that happens to refer to a terminal will cause the process to acquire a controlling terminal.
- In a containerized environment, a magic link inside */proc* might refer to an object outside the container and might provide a means to escape from the container.

Because of such risks, an application might prefer to disable magic link resolution by using the RESOLVE\_NO\_MAGICLINKS flag.

If the trailing component such as *basename* of *pathname* is a magic link, *how.resolve* contains RESOLVE\_NO\_MAGICLINKS and *how.flags* contains both O\_PATH and O\_NOFOLLOW, then an O\_PATH file descriptor referencing the magic link is returned.

**RESOLVE\_NO\_SYMLINKS**

Disallows resolution of symbolic links during path resolution. This option implies RESOLVE\_NO\_MAGICLINKS. If the trailing component (for example, the *basename*) of *pathname* is a symbolic link, *how.resolve* contains RESOLVE\_NO\_SYMLINKS and *how.flags* contains both O\_PATH and O\_NOFOLLOW, then an O\_PATH file descriptor that references the symbolic link is returned.



The effect of the `RESOLVE_NO_SYMLINKS` flag, which affects the treatment of symbolic links in all of the components of *pathname*, differs from the effect of the `O_NOFOLLOW` file creation flag in `how.flags`. This difference affects the handling of symbolic links only in the final component of *pathname*.

**Tip:** Because symbolic links are widely used, applications that use the `RESOLVE_NO_SYMLINKS` flag are encouraged to make its use configurable unless the flag it is used for a specific security purpose. Setting this flag indiscriminately (for example, for purposes that are not related to security) for all uses of `openat2()` might result in spurious errors on previously functional systems. For example, errors might occur if a system *pathname* that is used by an application in a new distribution release so that a *pathname* component now contains a symbolic link.

#### **RESOLVE\_NO\_XDEV**

Disallows traversal of mount points during path resolution, including all bind mounts. *pathname* must either be on the same mount as the directory referred to by *dirfd*, or on the same mount as the current working directory if *dirfd* is specified as `AT_FDCWD`.

**Tip:** Applications that use the `RESOLVE_NO_XDEV` flag are encouraged to make its use configurable unless it is used for a specific security purpose because bind mounts are widely used by users. Setting this flag indiscriminately (such as for purposes that are not related to security) for all uses of `openat2()` might result in spurious errors on previously functional systems. For example, errors might occur if a system *pathname* that is used by an application is modified in a new distribution release so that a *pathname* component now contains a bind mount.

#### **RESOLVE\_CACHED**

Fails the open operation unless all path components are present in the kernel's lookup cache. If any kind of revalidation or I/O is needed to satisfy the lookup, `openat2()` fails with the error `EAGAIN`. This flag is useful in providing a fast-path open that can be performed without resorting to thread offload or other mechanisms that an application might use to offload slower operations.

If other bits are set in `how.resolve`, an error is returned.

### **Related services**

- [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset” on page 406](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)

### **Characteristics and restrictions**

None.

### **Examples**

See [“BPX1OPN \(open\) example” on page 1267](#) and [“BPX4OPN \(open\) example” on page 1359](#).

## **opendir (BPX1OPD, BPX4OPD) — Open a directory**

---

### **Function**

The `opendir` callable service opens a directory so that it can be read with the `readdir` service.

### **Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN

Operation	Environment
<b>AMODE (BPX1OPD)</b>	31-bit
<b>AMODE (BPX4OPD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1OPD,(Directory_name_length,
              Directory_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4OPD with the same parameters.

## Parameters

### Directory\_name\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the name of the directory.

### Directory\_name

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Directory\_name\_length.

The name of a field that contains the name of the directory. The length of this field is specified in Directory\_name\_length.

### Return\_value

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the opendir service stores a directory file descriptor that describes the specified directory, if the request is successful; or -1, if it is not successful.

### Return\_code

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the opendir service stores the return code. The opendir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The opendir service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process does not have permission to search some component of the name that is specified as Directory_name; or it does not have permission to work with the directory itself.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Directory_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Directory_name.
<b>EMFILE</b>	Too many other files are already open for the process.
<b>ENAMETOOLONG</b>	Directory_name is longer than 1023 bytes; or a component of the path name is more than 255 bytes long.
<b>ENFILE</b>	Too many files are already open.
<b>ENOENT</b>	The specified directory was not found. The following reason codes can accompany the return code: JROpenDirNotFound and JRQuiescing.
<b>ENOTDIR</b>	Some component of the pathname is not a directory. The following reason code can accompany the return code: JRTargetNotDir.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the opendir service stores the reason code. The opendir service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The opendir service opens a directory so that the first readdir service starts reading at the first entry in the directory. For information about the readdir service, see [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory”](#) on page 656.
2. Return\_value is a file descriptor for a directory only. It can be used as *input* to services that expect a directory file descriptor. These services include closedir, rewinddir, and readdir.

**Related services**

- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory”](#) on page 124
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory”](#) on page 656
- [“rewinddir \(BPX1RWD, BPX4RWD\) — Reposition a directory stream to the beginning”](#) on page 701

## Characteristics and restrictions

None.

## Examples

See “BPX1OPD (opendir) example” on page 1267.

# openstat (BPX2OPN, BPX4OPS) – Open a file and obtain status information

---

## Function

The **openstat** callable service gains access to a file, creates a file descriptor for it, and obtains its status.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX2OPN)</b>	31-bit
<b>AMODE (BPX4OPS)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX2OPN, (Pathname_length,
               Pathname,
               Options,
               Mode,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4OPS with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Pathname of the file that is to be opened.

**Pathname**

Supplied parameter

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the Pathname\_length parameter.

The name of a field that contains the name of the file that is to be opened. The length of this field is specified in Pathname\_length.

Pathnames can begin with or without a slash:

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory. The search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Options**

Supplied parameter

**Type**

Structure

**Length**

Fullword

The name of a fullword that contains the binary flags that describe how the file is to be opened. For a description of the options, see the options for the open callable service in [“Usage notes” on page 517](#).

Options are mapped by the BPXYOPNF macro; see [“BPXYOPNF — Map flag values for open” on page 1113](#).

**Mode**

Supplied parameter

**Type**

Structure

**Length**

Fullword

The name of a fullword in which the mode field is specified. The mode field, which is mapped by BPXYMODE, specifies the file type and the permissions that are granted by the caller to itself, to its groups, and to any user. See [“BPXYFTYP — File type definitions” on page 1078](#).

**Status\_area\_length**

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of the fullword that contains the length of the Status\_area parameter. To determine the value of Status\_area\_length, use the BPXYSTAT macro (see [“BPXYSTAT — Map the response structure for stat” on page 1162](#)).

**Status\_area**

Supplied and returned parameter

**Type**

Structure

**Length**

The length of BPXSTAT or Status\_area\_length, whichever is less.

The name of an area to which the **openstat** service returns the status information for the file. Status\_area is mapped by the BPXSTAT macro (see [“BPXSTAT — Map the response structure for stat”](#) on page 1162).

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **openstat** service stores the file descriptor, if the file is opened successfully; or -1, if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **openstat** service stores the return code. The **openstat** service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The following table lists the values that are returned in the Return\_code parameter.

Return_code	Explanation
<b>EACCES</b>	The calling process was denied access for one of the following reasons: <ul style="list-style-type: none"> <li>• The calling process does not have permission to search one of the directories that was specified in the Pathname parameter.</li> <li>• The calling process does not have permission to open the file in the way that was specified on the Options parameter.</li> <li>• The file does not exist, and the calling process does not have permission to write into files in the directory in which the file would have been created.</li> <li>• The truncate option was specified, but the calling process does not have write permission for the file.</li> </ul>
<b>EAGAIN</b>	Resources were temporarily unavailable.
<b>EBUSY</b>	The Pathname parameter specified a manager pseudoterminal that is already in use, or for which the corresponding subsidiary pseudoterminal is open.
<b>EEXIST</b>	The exclusive create option was specified, but the file already exists. The following reason code can accompany the return code: JRFileExistsExclFlagSet.
<b>EFBIG</b>	A request to create a new file is prohibited because the file size limit for the process is set to 0.
<b>EINTR</b>	The <b>openstat</b> operation was interrupted by a signal.

Return_code	Explanation
<b>EINVAL</b>	The Options parameter does not specify a valid combination of the O_RDONLY, O_WRONLY and O_TRUNC bits; or the file type that was specified in the Mode parameter is not valid. The following reason codes can accompany the return code: JRInvOpenFlags and JROpenFlagConflict.
<b>EISDIR</b>	The file that is specified by Pathname is a directory, and the Options parameter specifies write or read/write access. The following reason code can accompany the return code: JRDirWriteRequest.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>EMFILE</b>	The process has reached the maximum number of file descriptors it can have open.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters. Filename truncation is not supported.
<b>ENFILE</b>	The maximum number of file descriptors that can be open has been reached.
<b>ENODEV</b>	<p>Typical causes:</p> <ul style="list-style-type: none"> <li>• An attempt was made to open a character special file for a device that is not supported.</li> <li>• An attempt was made to open a character special file for a device that is not yet initialized.</li> </ul> <p>The following reason code can accompany the return code: JRNoCTTY.</p>
<b>ENOENT</b>	<p>Typical causes:</p> <ul style="list-style-type: none"> <li>• The <b>openstat</b> request did not specify that the file was to be created, but the file that was named by Pathname was not found.</li> <li>• The <b>openstat</b> request specified that the file was to be created, but some component of Pathname was not found, or the Pathname parameter was blank.</li> </ul> <p>The following reason codes can accompany the return code: JREndingSlashOCreat, JRNoFileNoCreatFlag, and JRQuiescing.</p>
<b>ENOSPC</b>	The directory or file system that was intended to hold a new file has insufficient space.
<b>ENOTDIR</b>	A component of Pathname is not a directory.
<b>ENXIO</b>	The <b>openstat</b> request specified write-only and nonblock for a FIFO special file, but no process has the file open for reading. For pseudoterminals, this could mean that the minor number that is associated with the path name is too large.
<b>EPERM</b>	The caller is not permitted to open the specified subsidiary pseudoterminal; or the corresponding manager pseudoterminal is not yet open. EPERM is also returned if the subsidiary pseudoterminal is closed with HUPCL set and an attempt is made to reopen it.
<b>EROFS</b>	The Pathname parameter names a file on a read-only file system, but options that would allow the file to be altered were specified: write-only, read/write, truncate, or (for a new file) create. The following reason codes can accompany the return code: JRReadOnlyFileSetWriteReq and JRReadOnlyFileSetCreatReq.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **openstat** service stores the reason code. The **openstat** service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

For information about opening and creating files, see [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511.

**Related services**

- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ”](#) on page 238

**Characteristics and restrictions**

See the usage notes.

**Examples**

See [“BPX2OPN \(openstat\) example”](#) on page 1267.

## **\_\_passwd, \_\_passwd\_\_applid (BPX1PWD, BPX4PWD) — Verify or change security information**

---

**Function**

The \_\_passwd callable service verifies or changes the input user\_name's password or password phrase, or verifies the input user\_name's PassTicket.

**Requirements**

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key.
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PWD)</b>	31-bit
<b>AMODE (BPX4PWD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.



## Format

The syntax format is as follows:

```
CALL BPX1PWD,(User_name_length,
              User_name,
              Pass_length,
              Pass,
              New_Pass_length,
              New_Pass,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PWD with the same parameters.

## Parameters

### User\_name\_length

Supplied parameter .

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the length of User\_name.

### User\_name

Supplied parameter.

#### Type

Character string.

#### Character set

No restriction.

#### Length

Specified by the User\_name\_length parameter.

The name of a field, of length User\_name\_length, that contains, left-aligned, the name of the user whose Pass value is to be verified or changed.

### Pass\_length

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket or between 9 and 100 characters for a password phrase. A length of zero indicates that the Pass parameter is to be ignored and causes a SURROGAT class check.

See [“Usage notes” on page 534](#).

### Pass

Supplied parameter.

#### Type

Character string.

#### Character set

No restriction.

#### Length

Specified by the Pass\_length parameter.

**\_\_passwd (BPX1PWD, BPX4PWD)**

The name of a field, of length Pass\_length, that contains, left-aligned, the password, PassTicket or password phrase that is to be verified.

**New\_Pass\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the length of New\_Pass. This length must be between 1 and 8 characters for a password or between 9 and 100 characters for a password phrase. A length of zero indicates that New\_Pass is to be ignored.

**New\_Pass**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the New\_Pass\_length parameter

The name of a field, of length New\_Pass\_length, that contains, left-aligned, the new password or password phrase for the specified user.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the \_\_passwd service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the \_\_passwd service stores the return code. The \_\_passwd service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The \_\_passwd service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	User_name, Pass, or New_Pass length is incorrect; or the user name has an illegal first character. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRUserNameLenError, JRPASSWORDLenError, JRNewPasswordLenError, and JRUserNameBad.
<b>ESRCH</b>	The user name specified is not defined to OMVS.
<b>EACCES</b>	The password specified is not authorized; access is denied.

Return_code	Explanation
<b>EMVSERR</b>	There is an error in the USER definition in the security product data base. The following reason codes can accompany the return code: JREnvDirty, JRPNoSAFUser, JRSAFGroupNoOMVS, JRSAFUserNoOMVS, and JRSAFNoGid.  The caller environment is dirty; that is, a program was loaded from an unauthorized library.
<b>EMVSEXPIRE</b>	The password has expired.
<b>EMVSPASSWORD</b>	The new password is not valid.
<b>EMVSSAFEXTRERR</b>	A RACF authorization error has occurred. The reason code contains the RACF return and reason codes, respectively, in the two low-order bytes. See <a href="#">Table 13 on page 533</a> for more information.
<b>EMVSSAF2ERR</b>	A RACF authorization occurred. The reason code contains the RACF return and reason codes, respectively, in the two low-order bytes. See <a href="#">Table 13 on page 533</a> for more information.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the \_\_passwd service stores the reason code. The \_\_passwd service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

Table 13. RACF return and reason codes

Return code	Reason code	Explanation
<b>8</b>	12	Internal error during RACF processing
<b>8</b>	16	Unable to establish recovery
<b>8</b>	20	The user does not have appropriate RACF access to either the SECLABEL, SERVAUTH profile, or APPL.
<b>30</b>	00	The user is not authorized to the port of entry.
<b>30</b>	04	The user is not authorized to access the system on this day, or at this time of day.
<b>30</b>	08	The port of entry cannot be used on this day, or at this time of day.
<b>34</b>	N/A	The user is not authorized to use the application.
<b>38</b>	04	MLACTIVE requires a SECLABEL; none was specified.
<b>38</b>	08	The user is not authorized to the SECLABEL.
<b>38</b>	0C	The system was in a multilevel secure status, and the dominance check failed.
<b>38</b>	10	Neither the user's nor the submitter's security label dominates. They are disjoint.
<b>38</b>	14	The client's security label is not equivalent to the server's security label.

1. All return and reason codes are in hexadecimal.
2. Return codes 30, 34 and 38 are associated with the RACF RACROUTE REQ=VERIFY macro.
3. Return code 8 is associated with the initACEE (IRRSIA00) RACF callable service.

Table 13 on page 533 is not a complete list of all possible RACF return code and reason code combinations. For RACF codes not listed here, see [initACEE \(IRRSIA00\) callable service](#) in *z/OS Security Server RACF Callable Services*.

## Usage notes

1. If a profile is defined in the FACILITY class protecting the BPX.DAEMON resource, all programs that are loaded into the caller's address space must be controlled programs by the installed security product (such as RACF). If the \_\_passwd service detects that a load of a non-program control was done, it fails with an errno of EMVSERR and an errnojr of JRENVDIRTY. For more information, see [Establishing the correct level of security for daemons in z/OS UNIX System Services Planning](#).
2. New\_Pass is ignored if New\_Pass\_length is 0. If New\_Pass is specified for a password, the length must be less than or equal to 8. Further installation requirements might apply; for example, the length might need to be a minimum of 6. For a password phrase, the length must be less than or equal to 100 and greater than or equal to 9. If the Return\_code indicates EMVSPASSWORD, the installation exit routine might have failed the request because the New\_Pass did not meet some installation standard. If no installation exit is installed on this system, RACF rejected the password.
3. Mixed case passwords and PassTickets are supported if the installed security product supports mixed case; otherwise, passwords and PassTickets are folded to uppercase. Non-graphic characters are always folded to blanks.

The contents of the password phrase string are passed unchanged to the installed security product.

4. If an entry for the specified User\_name is not found in the user database, or if the User\_name is not defined to the OMVS segment, an ESRCH error is returned.
5. If the caller of the \_\_passwd service has read access to the BPX.SRV.userid SURROGAT class profile, where userid is the user ID specified in the User\_name parameter, a null Pass value (Pass\_length set to zero) can be specified, and the \_\_passwd service will return a successful Return\_value. For more information about setting up SURROGAT profiles, see [Defining servers to process users without passwords in z/OS UNIX System Services Planning](#).

If, however, a New\_Pass is specified and Pass\_length is specified as 0, the \_\_passwd service fails with an EINVAL.

6. When no Pass value is specified, a SURROGAT class check is made, ensuring the caller has access to the profile BPX.SRV.userid (where userid is the value that is specified on the User\_Name parameter). If the userid portion of the profile name has blanks in it, then RACROUTE REQUEST=AUTH results in ABEND282 RC5C. The dump is suppressed and the request fails with a return code of EMVSSAF2ERR and a reason code of JrRACFBlankExists
7. The \_\_passwd\_applid() function is equivalent to \_\_passwd() with the added feature that it also allows an application identifier (applid) to be supplied. The applid is used to verify the user's authority to access the application. When a PassTicket is specified, the applid value is also used in conjunction with the USERID to verify the PassTicket.

If an application is not using the \_\_passwd\_applid() function but still wants to pass an applid to this service, the application can set the applid value in the BPXYTHLI.

- THLIEP\_FunctionCode is set with ThliEP\_ApplSet.
- THLIEP\_ApplidLen is set to the length of the APPLID. If this value is less than 1 or greater than 8, then the ThliEP\_APPLID value is ignored.
- ThliEP\_APPLID is set to the APPLID value.

If there is no applid value passed and the calling process has done a pthread\_security\_np() call, the applid value defaults to OMVSAPPL.

If there is no applid value passed and the calling process has NOT done a pthread\_security\_np() call, the applid defaults to a null value.

Some applications may need the applid to be specified as the JOBNAME. The application should set the ThliF2\_SetApp bit prior to calling the password service. When this bit is on, the password service uses the application JOBNAME as the applid value passed to the security product. This is honored only if the process has not done a pthread\_security\_np() call. Specification of the applid in the THLIEP\_APPLID field or via the \_\_passwd\_applid() call overrides the ThliF2\_SetApp setting.

8. Although z/OS UNIX System Services supports password phrases that are 9-100 characters in length, your installation, or the installed security product can have additional rules for password phrase lengths. Ask your security administrator or system programmer if any additional rules apply.
9. The \_\_passwd() service supports identity context references (ICR). When \_\_passwd() is used in conjunction with pthread\_security\_np(), an ICR can be used to authenticate a user and then create a thread-level security context (ACEE), similar to using a user ID and password. Because an ICR is a one-time-use entity, the \_\_passwd() and pthread\_security\_np() services must be called from the same thread and the ICR specified on pthread\_security\_np() must be identical to the ICR specified on the preceding \_\_passwd().
 

**Note:** An ICR is not a user ID and password. It is a reference in the local identity context cache.
10. If environment variable BPXK\_MIN\_PWFOLD=YES is set then non-graphic characters will not be changed to blanks before being passed to the security product. This behavior will exist for all threads in the process where the environment variable was set
11. If available, the Peer IP address is logged in the string data of SMF Type 80 records when a user is authenticated.

## Related services

- [“getpwnam \(BPX1GPN, BPX4GPN\) — Access the user database by user name ” on page 305](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1PWD \(\\_\\_passwd, \\_\\_passwd\\_\\_applid\) example” on page 1276](#).

# pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name

---

## Function

The pathconf callable service determines the current values of a configurable limit or option (variable) that is associated with a file or directory.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PCF):</b>	31-bit
<b>AMODE (BPX4PCF):</b>	64-bit

**Operation****ASC mode:****Interrupt status:****Locks:****Control parameters:****Environment**

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1PCF, (Pathname_length,
               Pathname,
               Name,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PCF with the same parameters.

**Parameters****Pathname\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Pathname parameter.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name that contains the path name of the file. The file has the length that is specified in Pathname\_length.

Path names can begin with or without a slash:

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Name**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword that contains a value that indicates the configurable limit or option (variable) that is to be returned in Return\_value. Use the BPXYPCF macro to specify which path name variable you want returned; see [“BPXYPCF — Command values for pathconf and pathconf” on page 1114](#).

Variable returned	Description
<b>PC_CHOWN_RESTRICTED</b>	Change ownership ( “chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory” on page 104) function is restricted to a process with appropriate privileges (see “Authorization” on page 8), and to changing the group ID (GID) of a file only to the effective group ID of the process or to one of its supplementary group IDs.
<b>PC_LINK_MAX</b>	Maximum value of a file's link count.
<b>PC_MAX_CANON</b>	Maximum number of bytes in a terminal canonical input line.
<b>PC_MAX_INPUT</b>	Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed as input before reading them.
<b>PC_NAME_MAX</b>	Maximum number of bytes in a filename (not a string length; count excludes a terminating null).
<b>PC_NO_TRUNC</b>	Path name components longer than 255 bytes generate an error.
<b>PATH_MAX</b>	Maximum number of bytes in a path name (not a string length; count excludes a terminating null).
<b>PIPE_BUF</b>	Maximum number of bytes that can be written atomically when writing to a pipe.
<b>_POSIX_VDISABLE</b>	Terminal special characters maintained by the system can be disabled using this character value. For information on querying and setting these special characters, see <a href="#">“tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal” on page 926</a> or <a href="#">“tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal” on page 937</a> .
<b>PC_ACL</b>	The security product supports access control lists.
<b>PC_ACL_ENTRIES_MAX</b>	The maximum number of entries that can be placed in an access control list for the specified file.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the pathconf service returns the current value of the path name variable corresponding to the Name specified, or -1 if not successful.

If the named path name variable does not have a limit for the specified file, then Return\_value is set to -1, and Return\_code and Reason\_code remain unchanged.

If \_POSIX\_CHOWN\_RESTRICTED is specified for Name, and \_POSIX\_CHOWN\_RESTRICTED is active, Return\_value is set to 1.

If \_POSIX\_CHOWN\_RESTRICTED is specified for Name, and \_POSIX\_CHOWN\_RESTRICTED is not active, Return\_value is set to 0.

If `_POSIX_NO_TRUNC` is specified for Name, and `_POSIX_NO_TRUNC` is active, Return\_value is set to 1.

If `_POSIX_NO_TRUNC` is specified for Name, and `_POSIX_NO_TRUNC` is not active, Return\_value is set to 0.

If `PC_ACL` is specified for Name, and `PC_ACL` is supported, Return\_value is set to 1.

If `PC_ACL` is specified for Name, and `PC_ACL` is not supported, Return\_value is set to 0.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pathconf service stores the return code. The pathconf service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

If the named path name variable does not have a limit for the specified file, Return\_value is -1 and Return\_code is unchanged. Otherwise the pathconf service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	Search permission is denied for a component of the path prefix.
<b>EINVAL</b>	See the usage notes for situations in which EINVAL is returned. The following reason code can accompany the return code: JRNotSupportedForFileType.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters; or some component of the path name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	The named file does not exist; or the Pathname argument points to an empty string. The following reason code can accompany the return code: JRNotSupportedForFileType.
<b>ENOTDIR</b>	A component of the path prefix is not a directory.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pathconf service stores the reason code. The pathconf service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If Name refers to `MAX_CANON`, `MAX_INPUT`, or `_POSIX_VDISABLE`, the following applies:
  - If Pathname does not refer to a terminal file, the function returns -1 in Return\_value, and sets Return\_code to EINVAL.



2. If Name refers to NAME\_MAX, PATH\_MAX, or \_POSIX\_NO\_TRUNC, the following applies:
  - If Pathname does not refer to a directory, the function still returns the requested information using the parent directory of the specified file.
3. If Name refers to PC\_PIPE\_BUF, the following applies:
  - If Pathname refers to a pipe or a FIFO, the value that is returned applies to the referred to object itself. If Pathname refers to a directory, the value that is returned applies to any FIFOs that exist or that can be created within the directory. If Pathname refers to any other type of file, the pathconf service returns -1 in Return\_value, and sets the Return\_code to EINVAL.
4. If Name refers to PC\_LINK\_MAX, the following applies:
  - If File\_descriptor refers to a directory, the value that is returned applies to the directory.

## Related services

- [“fpathconf \(BPX1FPC, BPX4FPC\) — Determine configurable path name variables” on page 233](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1PCF \(pathconf\) example” on page 1269](#).

# pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal

## Function

The pause service suspends execution of the calling thread until delivery of a signal whose action is either to execute a signal-catching function or to end the thread.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PAS):</b>	31-bit
<b>AMODE (BPX4PAS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PAS,(Return_value,  
             Return_code,  
             Reason_code)
```

AMODE 64 callers use BPX4PAS with the same parameters.

## Parameters

### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pause service returns -1 if completion of a signal-handling function causes control to be returned. The pause service does not otherwise return to its caller.

### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pause service stores the return code. The pause service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pause service can return the following value in the Return\_code parameter:

Return_code	Explanation
EINTR	A signal was received and handled successfully.

### Reason\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pause service stores the reason code. The pause service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. A thread that calls the pause service does not resume processing until a signal is delivered with an action to either process a signal-handling function or end the thread. Some signals can be blocked by the thread's signal mask; see [sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask](#) on page 845 for details.
2. If an incoming unblocked signal ends the thread, pause never returns to the caller.
3. If the signal action is to process a signal-catching function, the signal interface routine (SIR), which is defined by the mvssigsetup call, is given control when the pause service returns.
4. A return code is set when any failures are encountered that prevent this function from completing successfully.

5. The signal interface routine is given control only when the PSW key of the caller is equal to the signal delivery key of the process. The signal delivery key is set to the PSW key of caller of the first callable service that dubbed the process.
6. If the caller has a PSW key that is different from the signal delivery key, or has a PSW key of zero, pause returns a return code of EMVSERR and a reason code of JRPSWKeyNotValid.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered” on page 851](#)
- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)

## Characteristics and restrictions

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413.](#)

## Examples

See [“BPX1PAS \(pause\) example” on page 1269.](#)

# pfsctl (BPX1PCT, BPX4PCT) — Physical file system control

---

## Function

The pfsctl callable service sends a command and argument to a physical file system (PFS). The meanings of the command and argument are specific to the PFS and are defined by the PFS.

For detailed information about the use of pfsctl, see [Using pfsctl \(BPX1PCT\) physical file system for z/OS UNIX in z/OS DFSMSdfp Advanced Services](#). For information about the zFS-specific pfsctl functions, see [pfsctl \(BPX1PCT\) in z/OS File System Administration](#).

Some of the pfsctl functions are defined in the BPXYCONS macro. For more information about the macro, see [“BPXYCONS — Constants used by services” on page 1062.](#)

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB. If SRB, AF_INET/AF_INET6 socket support only
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PCT)</b>	31-bit task or SRB mode
<b>AMODE (BPX4PCT)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PCT,(File_System_type,  
              Command,  
              Argument_length,  
              Argument,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4PCT with the same parameters.

## Parameters

### File\_System\_type

Supplied parameter

#### Type

Character string

#### Character set

Printable characters

#### Length

8 bytes

The name of a field that contains the 8-character name for the file system type. The name matches the TYPE operand that was specified on the FILESYSTYPE statement, or the NAME operand of the SUBFILESYSTYPE statement that defined this physical file system in the BPXPRMxx parmlib member.

### Command

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the command that is to be passed to the physical file system.

### Argument\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Argument parameter.

### Argument

Parameter supplied and returned

#### Type

Defined by the physical file system

#### Character set

No restriction

#### Length

Specified by the Argument\_length parameter

Specifies the name of a buffer, of length Argument\_Length, that contains the argument that is to be passed to the physical file system.

The buffer can be modified by the physical file system to return information to the caller.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pfsctl service returns -1 if the request is not successful.

Depending on the physical file system and the request involved, the length of any returned information that is placed in the Argument buffer may be returned here.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pfsctl service stores the return code. The pfsctl service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pfsctl service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EFAULT</b>	The Argument buffer address is not valid; or an address passed in the buffer is not valid.
<b>EINTR</b>	The service was interrupted by a signal.
<b>EINVAL</b>	A supplied parameter is incorrect.  One of the following Reason_codes may accompany this Return_code: <ul style="list-style-type: none"> <li>• JRFilesysNotThere - The File_System_type specified does not exist.</li> <li>• JrIOBufLengthInvalid - The Argument_length specified an incorrect value.</li> <li>• JrInvlOctlCmd - The Command value was negative.</li> </ul>
<b>EMVSPARM</b>	The command or argument parameters were rejected by the physical file system. In this case, the accompanying Reason_code is generated by the physical file system. Refer to its documentation to determine the exact reason that the error occurred.
<b>ENOSYS</b>	This function is not supported by the physical file system that was specified. The following reason code can accompany this return code: JRPfsctl.
<b>EPERM</b>	Permission was denied by the physical file system. The calling program does not have sufficient authority for the service that was requested.
<b>EIBMBADTCPNAME</b>	PC#SetIbmOptCmd was used, and the name that was specified did not match any of the transports configured under Common INET. The caller did not succeed in getting affinity to a single transport, and this is probably an error for the application.
<b>ENXIO</b>	PC#SetIbmOptCmd was used. The name that was specified did not match a socket stack, but Common INET is not configured on this system. Because this system does not have multiple socket transports configured, there is already a natural affinity to one single stack, and this failure may not be a problem for the application.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pfsctl service stores the reason code. The pfsctl service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. This service is provided for generic communication between a program that is running in a user process and a physical file system.

It is similar to `w_iocctl`, but the command is directed to the physical file system itself, rather than to or for a particular file or device.

2. There is no restriction on the length of the argument buffer. The address and length of the argument buffer are passed to the physical file system in a UIO structure on the `vfs_pfsctl` operation.
3. See [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041 for more information about programming considerations for SRB mode.
4. As an example of how a physical file system writer might make use of this function, consider the requirement to display status and performance statistics about the physical file system. You can collect this information in the physical file system, but you need a way to display it to the user.

With pfsctl, your status utility program can easily fetch the information that it needs from the physical file system. The utility needs to know the File\_System\_type name that the physical file system was started with, and this can be made known to it by the physical file system with the Name/token callable services. For information about the Name/token callable services, see [Sharing application data \(name/token callable services\)](#) in *z/OS MVS Programming: Authorized Assembler Services Guide*.

5. Command values less than X'0x40000000' are considered to be authorized commands. A check for appropriate privileges (see “Authorization” on page 8) is made and the results of this check are passed to the physical file system in the `osi_privileged` bit.
6. Command values less than zero are reserved by the system.
7. **PC#SetIbmOptCmd** — This pfsctl service chooses a particular sockets transport; this is similar to the `setibmopt(IBMTCPIP_IMAGE)` C function.

The Command value for this function is X'C0000005'. You specify the transport with the File\_System\_type parameter. The Argument parameter is not used, and Argument\_length should be 0, unless you are setting persistent address space affinity.

The PC#SetIbmOptCmd function is used by programs that must connect to a specific socket transport, when z/OS UNIX is configured with multiple transports for the AF\_INET or AF\_INET6 address families. The specific socket transport is also known as a specific TCP/IP stack. After a transport is chosen, all subsequent socket requests for address family AF\_INET or AF\_INET6 create sockets that are exclusively attached to that single transport.

This is similar to the function provided by `ioctl(SIOCSETRTTD)`, except that `ioctl(SIOCSETRTTD)` detaches an existing socket from all but the specified transport, while `pfsctl(PC#SetIbmOptCmd)` causes future sockets to be attached to only the one transport. Using pfsctl for this function is significantly more efficient than using ioctl.

When there is only one transport that is configured, all socket requests for that address family go directly to it, regardless of any prior calls to `pfsctl(PC#SetIbmOptCmd)`. A call to `pfsctl(PC#SetIbmOptCmd)` is therefore not necessary in a single transport configuration, but the call will still fail if the name that is specified does not match that of a socket stack. There could be something wrong in the caller's configuration files that need to be addressed.

A pfsctl(PC#SetIbmOptCmd) request can be issued more than once to change the chosen transport and affect future sockets that are created. If File\_System\_type is all blanks, the caller's process is reset to indicate no transports chosen.

The chosen transport is inherited over fork and preserved over exec. If this is not desired, the child process should call pfsctl(PC#SetIbmOptCmd) with a blank name to reset itself.

If Argument\_length is four and the Argument value is one, this transport affinity also becomes an address space-level transport affinity. Otherwise, only process-level affinity is established. Address-space affinity persists over job steps within a job and over UNIX process termination and redubbing in that address space. It applies to all UNIX processes running within that address space, so long as the MVS JOBID of the address space does not change. Clearing a process's affinity also clears the address space affinity if an argument value of one is passed on that call.

Address-space-level affinity is intended for multiple job-step procedures in which one job step makes a call to pfsctl so that a program in a later job step will be restricted to the one specified transport (where that program does not have its own call to pfsctl or cannot be changed to do so). It can also be used to set affinity for a TSO address space, which affects all the programs and commands invoked afterward.

To minimize the performance impact of this feature, an address space is checked for address-space level affinity only once in the life of a process, and that check is only made in the socket, gethostid, and gethostname functions. Consequently, the effect of setting address space affinity when other processes are running in the address space, or for future programs that have their own calls to pfsctl, is unpredictable. Address-space level affinity is not, strictly speaking, inherited over fork; however, it is applied to a process the first time a call to socket, gethostid, or gethostname is made, so that if the fork occurs after one of those calls, the process's affinity is inherited by its children.

The BPXTCAFF program that is supplied by IBM can also be used to establish an address-space-level transport affinity for started procedures, submitted job streams, and the TSO CALL command. The BPXTCAFF program takes one parameter, the transport name, and makes a call to pfsctl(PC#SetIbmOptCmd), passing that name with an argument value of one, as follows:

```
//STEP0 EXEC,PGM=BPXTCAFF,PARM=TPNAME
```

For more information about transport affinity, see [Using specific transports under CINET in z/OS UNIX System Services Planning](#).

**8. PC#SetIbmAsyIO** — This pfsctl service chooses a Sockets transport that supports asynchronous I/O.

The Command value for this function is X'C0000006'. The File\_System\_type and Argument parameters are not significant, and Argument\_length should be 0.

This is similar to the function provided by PC#SetIbmOptCmd, except that you do not have to know the name of the TCP/IP stack.

**Note:** This function is obsolete. Do not use it.

When there is only one transport, all socket requests for that address family go directly to it, regardless of any prior calls to pfsctl(PC#SetIbmAsyIO). It is not an error to call pfsctl(PC#SetIbmAsyIO) when there is only one transport configured, therefore, programs using this function do not have to be sensitive to how an installation is configured. If the single transport does not support asynchronous I/O, attempts to call asyncio later will fail.

The choice of an asynchronous capable transport can be reset with a call to pfsctl(PC#SetIbmOptCmd) with a File\_System\_type of all blanks.

The chosen transport is inherited over fork and preserved over exec. If this is not desired, the child process should call pfsctl(PC#SetIbmOptCmd) with a blank name to reset itself.

**9. PC#ErrorText** — This pfsctl command retrieves error text for z/OS UNIX return codes and reason codes and for TCP/IP and zFS reason codes. You can use this service to request the error description for a specified return code or the error description, action text, and issuing module name for a specified reason code.

The Command value for this function is X'C000000B'.

On entry, the Argument parameter specifies the buffer in which to pass the request type and the return code or reason code and to receive the requested text. The buffer header contains fields to pass the type of text being requested, the type of error code (return code or reason code), and the return code or reason code and is mapped as follows:

Offset	Length	Field
0	2	Text request type
2	2	Error code type
4	4	Reason code (for PC#EtReason)
6	2	Return code (for PC#EtErrno)

The following request types indicate the type of text to be returned:

Text request type	Value	Text request description
PC#EtDesc	X'0000'	Get the error description text for a return code or reason code
PC#EtAction	X'0001'	Get the action text for a reason code
PC#EtModname	X'0002'	Get the issuing module name for a reason code

The following error code types indicate whether the error code you are passing is a return code or a reason code:

Error code type	Value	Error code description
PC#EtReason	X'0000'	Request is for a reason code
PC#EtErrno	X'0001'	Request is for a return code

The Argument\_length includes the length of the buffer header and must be large enough to receive the requested text.

On return, the requested text starts at the beginning of the buffer, overlaying the header. The Return\_value indicates the number of bytes returned in the buffer. If the buffer is not long enough to hold all of the requested text, the service only returns the amount of data that fits in the buffer; there is no explicit indication that data was truncated.

10. **PC#TDNames** — This pfsctl service returns a list of the names of all the transport stacks that are configured under Common INET. These are the names specified with the NAME() parameter of the SUBFILESYSTYPE statements from the BPXPRMxx parmlib member.

The Command value for this function is X'C000000F'.

The output of this function is a simple array of 8-byte names that are left justified and padded with blanks. The Argument parameter, which is used for the output area, must be large enough to hold all the names. If the Argument parameter is not large enough, it is filled with the number of whole names that will fit, and no indication is given that there are more names.

The maximum number of stacks configurable under CINET is 32, so an argument that is 256 bytes long will always be large enough.

The Return\_value indicates the number of names that have been returned. If CINET is not configured, the Return\_value will be zero.

The File\_System\_type parameter is not used with this command.

11. **PC#DirGetHost** — This pfsctl service can be used to direct a BPX1HST gethostid() or gethostname() request to a particular socket transport. The PC#DirGetHost function can be used by programs that need to get the host ID or host name of a specific TCP/IP stack without setting or changing the stack affinity for a process.



The Command value for this function is 'C0000014'x.

You use the File\_System\_type parameter to pass the desired transport and the Argument parameter to pass the BPX1HST request parameters. The Argument is mapped as follows:

Offset	Length	Field
0	4	Input: Domain value
4	4	Input: 0 for gethostid or Length of the Name area
8	*	Output: Name area for gethostname

The Argument\_length is the total length of the structure that is being passed, including the space reserved for the output Name on a gethostname() request. Refer to BPX1HST for details about the parameters and the difference between the gethostid() and gethostname() requests.

When Common Inet is not configured, the transport name is ignored and the BPX1HST request is directed to the one and only transport.

The output from a successful call to pfsctl will be the output that would have been received from BPX1HST, as follows:

- For gethostid() requests, the Return\_value parameter contains the ID.
- For gethostname() requests, the PC\_Name field of the PC\_DirGetHostArg structure contains the name.

For more information about transport affinity, see [Using specific transports under CINET in z/OS UNIX System Services Planning](#).

12. **PC#IsSrcAddr** — This pfsctl service is used to validate the passed source IP address against a set of input source address selection preference flags.

The Command value for this function is 'C0000018'x.

In a Common Inet configuration the File\_System\_Type parameter can be used to direct the request to a specific TCPIP stack. This will override any process stack affinity that might be established.

When File\_System\_Type is all blanks or all zeros the request is routed to the TCPIP stack specified in the scope\_id of the sockaddr or to the stack that “owns” the provided source IP address. If the provided IP address is not in the home address list of any active TCPIP stack, then the request will be routed to the default TCPIP stack.

When Common Inet is not configured the File\_System\_Type parameter is ignored.

Argument value is the IsSrcAddr structure which is defined in the BPXYSOCK macro. This function can be invoked using the inet6\_is\_srcaddr() C function.

Return Value	Explanation
0	The input address does not match an address that satisfies the source address selection preference flags indicated.
1	The IPv6 address corresponds to a valid address in the node and satisfies the given source address selection preference flags.
-1	The IPv6 address input value does not correspond to any address in the node or the source address selection flags are not one of the valid preference flags or the PFSCCTL service failed.

This PFSCCTL service is only routed to a single active IPv6 TCPIP stack and not routed by the CINET on failure.

This service is only available for IPv6 addresses and IPv4-mapped IPv6 addresses. If an AF\_INET6 NETWORK statement is not specified in the BPXPRMxx parmlib member or at least one AF\_INET6 supported TCPIP stacks is not active, then the service is rejected with return code and reason codes.

## Characteristics and restrictions

None.

## Examples

See “BPX1PCT (pfsctl) example” on page 1269.

# \_\_pid\_affinity (BPX1PAF, BPX4PAF) – Add or delete an entry in a process's affinity list

---

## Function

The \_\_pid\_affinity service adds or deletes an entry in a process's affinity list. When a process terminates, each process in its affinity list is notified (sent a signal) of the event. The \_\_pid\_affinity service dynamically creates or breaks an association between two processes. Its function is similar to the notification mechanism between parent and child processes, except that in this case the processes are not related in any way.

## Requirements

### Operation

### Environment

#### Authorization:

Supervisor state or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1PAF):

31-bit

#### AMODE (BPX4PAF):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PAF,(Function_code,  
              Target_Pid,  
              Signal_Pid,  
              Signal,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4PAF with the same parameters.

## Parameters

### Function\_code

Supplied parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword that specifies a numeric value that identifies the function to be performed. The following Function\_code constants are defined by the BPXYCONS macro (see [“BPXYCONS — Constants used by services”](#) on page 1062):

**Constant**

**Function**

**PAF\_ADD\_PID#**

The process and associated signal that are specified by Signal\_Pid are to be added to the affinity list of the process that is specified by Target\_Pid.

**PAF\_DELETE\_PID#**

The process and associated signal that are specified by Signal\_Pid are to be deleted from the affinity list of the process that is specified by Target\_Pid.

**Target\_Pid**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that specifies a numeric value that identifies the PID (Process ID) of the process whose affinity list is to be altered. See the usage notes for limitations on the PIDs that can be specified.

**Signal\_Pid**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that specifies a numeric value that identifies the PID (Process ID) of the process that, when the Target\_Pid process terminates, is to be sent the signal that is specified by the Signal parameter. See the usage notes for limitations on the PIDs that can be specified.

**Signal**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that specifies a numeric value that identifies the signal that the Signal\_Pid process is to receive when the process that is specified by Target\_Pid terminates. The signal must be one that is defined by the BPXYSIGH macro.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_pid\_affinity service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_pid\_affinity service stores the return code. The \_\_pid\_affinity service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_pid\_affinity service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EINVAL**

One or more of the following conditions were detected:

- The value of Signal is not a supported signal.
- Target\_Pid does not contain a value greater than 1.
- Signal\_Pid does not contain a value greater than 1.
- Signal\_Pid and Target\_Pid are the same.

The following reason codes can accompany the return code: JRInvalidSignal, JRTargetPid, JRPidsSame, and JRSignalPid.

**EPERM**

The caller does not have permission to send a signal to the process that is specified on the Signal\_Pid parameter.

**EMVSSAF2ERR**

A System Authorization Facility (SAF) or RACF call had an error.

**ESRCH**

One or more of the following conditions were detected:

- No process was found that corresponds to Target\_Pid.
- No process was found that corresponds to Signal\_Pid.

The following reason codes can accompany the return code: JRTargetPid and JRSignalPid.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_pid\_affinity service stores the reason code. The \_\_pid\_affinity service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR, Reason\_code contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF Check Privilege service return and reason code values, see the following table:

**RACF return code   RACF reason code   Explanation**

<b>8</b>	<b>4</b>	The caller is not the owner of the target process.
<b>8</b>	<b>12</b>	There was an internal error during RACF processing.

## Usage notes

1. The PIDs that are specified by the Target\_Pid and Signal\_Pid parameters must be greater than 1. Specifying a PID that is equal to or less than 1 results in an error.
2. In order for the caller to add an entry to the affinity list of a process (Target\_Pid), the Signal\_Pid process must exist, and the caller's process must have permission to send it a signal.
3. During process termination, the process attempts to send all the specified signals to the corresponding PID or PIDs in its affinity list. If a signal cannot be sent (for instance, if the process has already terminated), termination continues.
4. If a process changes identity after it has been added to another process's affinity list, the signal is sent upon process termination without permission being reverified.
5. Identical entries that contain the same PID (Signal\_Pid) and signal are not allowed in a process's affinity list. If an attempt is made to add a process and an identical entry is found, the service completes successfully without adding another entry.
6. To delete an entry from an affinity list, the PID (Signal\_Pid) specified by the caller must be the same as an entry in the Target\_Pid process's affinity list.

## Related services

- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1PAF \(\\_\\_pid\\_affinity\) example” on page 1268](#).

# pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe

---

## Function

The pipe callable service creates a pipe. A pipe is an I/O channel that a process can use to communicate with another process, with another thread (in this same process or another process), or in some cases with itself. Data can be written into one end of the pipe and read from the other.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PIP):</b>	31-bit
<b>AMODE (BPX4PIP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1PIP,(Read_file_descriptor,  
              Write_file_descriptor,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4PIP with the same parameters.

Parameters

Read\_file\_descriptor

Returned parameter

Type:

Integer

Length:

Fullword

The name of a fullword in which the pipe service stores the file descriptor for the read end of the pipe if the pipe is created successfully.

Write\_file\_descriptor

Returned parameter

Type:

Integer

Length:

Fullword

The name of a fullword in which the pipe service stores the file descriptor for the write end of the pipe if the pipe is created successfully.

Return\_value

Returned parameter

Type:

Integer

Length:

Fullword

The name of a fullword in which the pipe service returns 0 if the request is successful, or -1 if it is not successful.

Return\_code

Returned parameter

Type:

Integer

Length:

Fullword

The name of a fullword in which the pipe service stores the return code. The pipe service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The pipe service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EMFILE	The process has either reached the maximum number of file descriptors it can have open, or the current pipe limit was exceeded. Refer to the reason code that was provided.

Return_code	Explanation
<b>ENFILE</b>	Opening the pipe would exceed the number of files that the system can have open simultaneously.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pipe service stores the reason code. The pipe service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Processes can read from the **Read\_file\_descriptor** and write to the **Write\_file\_descriptor**. Data written will be read first-in, first-out (FIFO).
2. When the pipe call creates a pipe, the O\_NONBLOCK and FD\_CLOEXEC flags are turned off on both ends of the pipe. You can turn on these flags with the fcntl call; see [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216.

**Related services**

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket”](#) on page 652
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket”](#) on page 1032

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PIP \(pipe\) example”](#) on page 1270.

## pipe2 (BPX1PI2, BPX4PI2) — Create a new pipe instance

---

**Function**

The **pipe2** service creates a new pipe instance and returns a file descriptor referring to that instance. The returned pipe instance can then be used to register interest in particular file descriptors using the pipe family of functions (READ,WRITE and CLOSE).

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN

<b>Operation</b>	<b>Environment</b>
<b>AMODE (BPX1PI2)</b>	31-bit
<b>AMODE (BPX4PI2)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PI2 (pipefd[0],
              pipefd[1],
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PI2 with the same parameters.

Parameters

pipefd[]

Supplied parameter.

Type

Unsigned 64-bit integer.

Length

Doubleword.

pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe.

Flags

Supplied parameter.

Type

Integer

Length

Fullword

The name of a fullword field that indicates the Initialization options.

Flag	Value	Description
<b>O_CLOEXEC</b>	'00001000' X	Sets the close-on-exec (FD_CLOEXEC) flag on the two new file descriptors



Flag	Value	Description
<b>O_DIRECT</b>	'00002000'X	<p>Create a pipe that performs I/O in packet mode. Each Write to the pipe is dealt with as a separate packet, and read from the pipe will read one packet at a time.</p> <ul style="list-style-type: none"> <li>Writes of greater than PIPE_BUF bytes are split into multiple packets. Use BPX1PCF to obtain the value for PIPE_BUF.</li> <li>If a read specifies a buffer size that is smaller than the next packet, then the requested number of bytes are read and the excess bytes in the packet are discarded. Specifying a buffer size of PIPE_BUF is sufficient to read the largest possible packets (see the preceding bullet).</li> <li>Zero-length packets are not supported. A read that specifies a buffer size of zero will return 0.</li> </ul>

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **pipe2** service returns the **pipe2** file descriptor that was created, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **pipe2** service stores the return code. The **pipe2** service returns the Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The **pipe2** service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	<i>pipefd</i> is not valid.
<b>EINFILE</b>	The Options parameter does not specify a valid value.
<b>EMFILE</b>	The process has either reached the maximum number of file descriptors that it can have open.
<b>ENFILE</b>	The maximum number of file descriptors that can be open has been reached.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pipe2 service stores the reason code. The pipe2 service returns the Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Processes can read from the **Read\_file\_descriptor** and write to the **Write\_file\_descriptor**. Data written will be read first-in, first-out (FIFO).
2. When the pipe call creates a pipe, the O\_NONBLOCK and FD\_CLOEXEC flags are turned off on both ends of the pipe. You can turn on these flags with the fcntl call; see [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216.

**Related services**

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket”](#) on page 652
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket”](#) on page 1032

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PI2 \(pipe2\) example”](#) on page 1270 and [“BPX4PI2 \(pipe2\) example”](#) on page 1362.

## **[pivot\\_root \(BPX1PVR, BPX4PVR\) — Change the root mount](#)**

---

**Function**

The pivot\_root system call changes the root mount in the mount namespace of the calling process. More precisely, it moves the root mount to the directory put\_old\_path and makes new\_root\_path the new root mount. Pivot\_root also changes the root directory and the current working directory of each process or thread in the same mount namespace to new\_root\_path if they point to the old root directory.

For the macro, see [“BPXYMNTE — Map response and element structure of w\\_getmntent”](#) on page 1104.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PVR)</b>	31-bit
<b>AMODE (BPX4PVR)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

```
CALL BPX1PVR(New_Root_Path_Length,
             New_Root_Path,
             Put_Old_Path_Length,
             Put_Old_Path,
             Return_Value,
             Return_Code,
             Reason_Code)
```

AMODE 64 callers use BPX4PVR with the same parameters.

**Parameters****New\_Root\_Path\_Length**

Supplied parameter.

**Type**

Integer

**Length**

Word

The name of a fullword field that contains the length of the New\_Root\_Path.

**New\_Root\_Path**

Supplied parameter.

**Type**

String

**Length**

New\_Root\_Path\_Length

The name of field that contains the path to a file name that represents the new\_root directory.

**Restriction:** New\_root\_Path must be a mount point directory.

**Put\_Old\_Path\_length**

Supplied parameter.

**Type**

Integer

**Length**

Word

The name of a fullword field that contains the length of the Put\_Old\_Path.

**Put\_Old\_Path**

Supplied parameter.

**Type**

String

**Length**

Put\_Old\_Path\_Length

The name of field that contains the path to a file name that represents the put\_old directory.

**Restrictions:**

- Put\_Old\_Path must be at or beneath New\_Root\_Path.

- Put\_Old\_Path cannot be a mount point, unless Put\_old\_path is equal to New\_Root\_Path.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pivot\_root service returns 0, if the request was successful or 1, if it was not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pivot\_root service stores the return code. The clone service sets the Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pivot\_root service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBUSY</b>	New_Root_Path or Put_Old_Path is on the current root mount.
<b>EINVAL</b>	New_Root_Path is not a mount point.
<b>EINVAL</b>	Pivot_Root cannot be called from the root namespace.
<b>EINVAL</b>	Put_Old_Path is not at or underneath New_Root_Path.
<b>EINVAL</b>	The current root directory is not a mount point.
<b>EINVAL</b>	Put_Old_Path is a mount point.
<b>ENOTDIR</b>	New_Root_Path or Put_Old_Path is not a directory.
<b>EPERM</b>	The calling process does not have authority.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

**Usage notes for pivot\_root**

1. A pivot\_root operation cannot be performed on the initial namespace.
2. New\_root\_Path and Put\_Old\_Path cannot be the same as the system root directory.
3. New\_root\_Path and Put\_Old\_Path can reference the same directory.

**Related services**

- [“clone \(BPX1CLN, BPX4CLN\) — Create a child process” on page 114](#)
- [“chroot \(BPX1CRT, BPX4CRT\) — Change the root directory” on page 111](#)
- [“unshare \(BPX1UNS, BPX4UNS\) — Run program in a new namespace” on page 977](#)

## Characteristics and restrictions

None.

## Examples

See “BPX1PVR (pivot\_root) example” on page 1275 and “BPX4PVR (pivot\_root) example” on page 1368.

# **\_\_poe() (BPX1POE, BPX4POE) – Port of entry information**

---

## Function

The \_\_poe() callable service specifies the port of entry information the system is to use in determining various levels of permission checking in a multilevel-secure system. The authorization that is required to invoke this service is one of the following:

- Read access to the BPX.POE resource in the FACILITY class
- A UID of 0 when the BPX.POE resource is not defined

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1POE)</b>	31-bit
<b>AMODE (BPX4POE)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1POE,(Poecb_length,
              Poecb,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4POE with the same parameters.

## Parameters

### Poecb\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword field that contains the length of the Poecb control block that is being passed in the next parameter. To determine the value of Poecb\_length, use the BPXYPOE macro ([“BPXYPOE — Map poe syscall parameters” on page 1123](#)).

**Poecb**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Specified by the Poecb\_length parameter

The name of a Poecb structure that is to be used to control this port of entry operation. See the section on the Poecb control block in the usage notes for details on setting the fields of the Poecb. The BPXYPOE macro ([“BPXYPOE — Map poe syscall parameters” on page 1123](#)) maps the Poecb.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_poe() service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_poe() service stores the return code. The \_\_poe() service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The \_\_poe() service returns one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	The Poecb structure is incorrect. Consult Reason_code to determine the exact reason the error occurred. The following reason codes unique to the __poe() service can accompany the return code: JRPoeLenErr, JRPoeScopeErr, JRPoeEntryTypeErr, and JRPoeEntryLenErr.
<b>EPERM</b>	The calling address space does not have the appropriate privileges to set the POE attributes.
<b>EFAULT</b>	A bad address was received in the POEEntryPtr field of the BPXYPOE mapping that was pointed to by the Poecb parameter.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_poe() service stores the reason code. The \_\_poe() service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The ability to register port of entry is a privileged operation. An installation has two ways of allowing an application to use this service:
  - a. For the highest level of security, the installation defines the BPX.POE resource in the FACILITY class. For an application to use this service, the user ID it runs under must be given read access to that resource.
  - b. For a lower security arrangement, you can assign the user ID under which the application runs a UID of 0 so that it operates as a superuser.
2. Poecb control block: When the POEEntryPtr field in the BPXYPOE mapping contains the address of a file descriptor, the caller must indicate in the POEEntryType field what type of file the file descriptor represents. The two supported file types are:
  - POE#EntrySocket — the file descriptor is for a socket file.
  - POE#EntryFile — the file descriptor is for a non-socket file. This includes the following file types:
    - Character special
    - FIFO
    - Regular
    - Symbolic link
    - Directory

To clear the POE data that has been registered for a thread or process, specify the following data and options in the poecb and call the \_\_poe() service.

- POE#Attributes set to zeros (X'00) or blanks (X'40)
- POE#ActionWrite
- POE#ScopeProcess or POE#ScopeThread

When the POE data for a process or thread has been cleared, that data will no longer be included in the search order that z/OS UNIX services use to determine the POE data to be passed to RACF when creating a security context. For example, when both process and thread POE data has been registered, the thread-level data would take precedence over the process data. Clearing the thread-level data would cause it to be skipped and the process level data would be used if available.

POE#ScopeThread, POE#ScopeProcess and POE#ScopeSocket are mutually exclusive; only one must be specified. If none or more than one are specified, the request will fail with -1/EINVAL/JRPoeScopeErr

POE#ReadPOE, POE#WritePOE and POE#SetGetPOE are mutually exclusive. If more than one is specified, the request will fail with EINVAL/JRPoeActionErr.

If the POE#ReadPOE, POE#WritePOE or POE#SetGetPOE options are specified, the storage obtained for the POE control block should be POE#LenV2 bytes in length. POE#LenV2 must be specified for the Poecb\_length parameter on the BPX1POE syscall. If the length is incorrect the request will fail with -1/EINVAL/JrPOELenErr

Table 14. Poecb control block: Socket scope

Action	Source	Destination	Results
<b>Read</b>	Socket or file descriptor	POEAttributes area in the BPXYPOE mapping	POE data is extracted from the file/socket descriptor supplied by the caller and returned to the caller via the BPXYPOE area. The caller's thread and process level POE data in the Oaob and OtcB is unchanged.
<b>Write</b>	Not applicable	Not applicable	Request fails with EINVAL/JrPoeSocketScopeErr
<b>SetGet</b>	Not applicable	Not applicable	Request fails with EINVAL/JrPoeSocketScopeErr
<b>None</b>	Not applicable	Not applicable	Request fails with EINVAL/JrPoeSocketScopeErr

Table 15. Poecb control block: Process scope

Action	Source	Destination	Results
<b>Read</b>	Socket or file descriptor	POEAttributes area in the BPXYPOE mapping	POE data is extracted from the file/socket descriptor supplied by the caller and returned to the caller via the BPXYPOE area. The caller's thread and process level POE data in the Oaob and OtcB is unchanged.
<b>Write</b>	n/a	n/a	Request fails with EINVAL/JrPoeSocketScopeErr
<b>SetGet</b>	n/a	n/a	Request fails with EINVAL/JrPoeSocketScopeErr
<b>None</b>	n/a	n/a	Request fails with EINVAL/JrPoeSocketScopeErr
<b>R</b>	Process level (OapbPOEAttr)	POEAttributes area in the BPXYPOE mapping	Process level POE data copied from the Oapb is returned to the caller via the BPXYPOE area
<b>W</b>	POEAttributes area in the BPXYPOE mapping	Process level (OapbPOEAttr)	POE data received from the caller in the BPXYPOE area is copied to the process level POE data in the Oapb
<b>SetGet</b>	Socket or file descriptor	Process level (OapbPOEAttr) and POEAttributes area in the BPXYPOE mapping	POE data is extracted from the file/socket descriptor supplied by the caller. The data is copied to the process level POE data in the Oapb and returned to the caller via the BPXYPOE area
<b>None</b>	Socket or file descriptor	Process level (OapbPOEAttr)	POE data is extracted from the file/socket descriptor supplied by the caller and copied to the process level POE data in the Oapb

Table 16. Poecb control block: Thread scope

Action	Source	Destination	Results
<b>R</b>	Thread level (OtcBPOEAttr)	POEAttributes area in the BPXYPOE mapping	Thread level POE data copied from the OtcB is returned to the caller via the BPXYPOE area



Table 16. Poecb control block: Thread scope (continued)			
Action	Source	Destination	Results
<b>W</b>	POEAttributes area in the BPXYPOE mapping	Thread level (OtcbPOEAttrs)	POE data received from the caller in the BPXYPOE area is copied to the thread-level POE data in the Otcb
<b>SetGet</b>	Socket or file descriptor	Thread level (OtcbPOEAttrs) and POEAttributes area in the BPXYPOE mapping	POE data is extracted from the file/socket descriptor supplied by the caller. The data is copied to the thread-level POE data in the Otcb and returned to the caller via the BPXYPOE area
<b>None</b>	Socket or file descriptor	Thread level (OtcbPOEAttrs)	POE data is extracted from the file/socket descriptor supplied by the caller and copied to the thread-level POE data in the Otcb

3. The POE data registered via the \_\_poe() service is passed to RACF by z/OS UNIX services that create a new security context (ACEE). The z/OS UNIX services that utilize POE data are:
- \_\_passwd()
  - pthread\_security\_np()
  - \_\_login()
  - setuid()/seteuid()/setreuid()
  - spawn (only with user ID change)
4. When z/OS UNIX services call RACF to build a new security context, POE data is passed to RACF using the parameters in [Table 17 on page 563](#) for RACROUTE REQUEST=VERIFY ENVIR=CREATE

Table 17. RACROUTE parameters for POE data	
POE data	RACROUTE parameter
<b>POELabel</b>	SECLABL
<b>POEProfile</b>	SERVAUTH
<b>POETermid</b>	TERMID. is only specified by the z/OS UNIX __passwd() and setuid()/seteuid()/setreuid() services.

The amount and source of POE data passed to RACF varies based on the caller's environment.

- When the caller's address space security label is non-SYSMULTI, only the SECLABL parameter is specified.
- When the caller's address space security label is SYSMULTI, the following RACROUTE parameters are specified when the corresponding POE data is found in the search order. POE data from only one level will ever be specified. When valid POE data is found in the search order the search stops. For example, if a thread-level POELabel (SECLABL) is found the search will continue for POEProfile (SERVAUTH) and POETermid (TERMID) thread-level data but will not search at the process or address space level.
  - SECLABL
    - a. Thread level (Otcb)
    - b. Process level (Oapb)
    - c. Address space seclabel (SYSMULTI)
  - SERVAUTH
    - a. Thread level (Otcb)
    - b. Process level (Oapb)

## poll (BPX1POL, BPX4POL)

- TERMID (only for the `__passwd()` and `setuid(0)/seteuid()/setreuid()` services)
  - a. Thread level (Otcb)
  - b. Process level (Oapb)
- 5. Table 18 on page 564 describes the POE data propagation for z/OS UNIX services that create a new process, a new process image (`exec`), or a new thread.

Table 18. POE data propagation for z/OS UNIX services		
Service	Thread POE data	Process POE data
<b>fork()</b>	Yes	Yes
<b>spawn()</b>	No	Yes
<b>exec()</b>	No	Yes
<b>pthread_create()</b>	No	Yes. * The newly created thread is in the same process as the thread calling <code>pthread_create()</code> . By default, process level POE data is shared between all thread in the process.
<b>attach_exec()</b>	No	Yes
<b>attach_execMVS()</b>	No	Yes
<b>execMVS()</b>	No	Yes

### Characteristics and restrictions

The `__poe()` service is restricted to users that have the appropriate privileges, as defined under “Function” on page 559.

### Examples

See “BPX1POE (`__poe`) example” on page 1270.

## poll (BPX1POL, BPX4POL) – Monitor activity on file descriptors and message queues

### Function

The poll service checks the I/O status of multiple open file descriptors and message queues. The file descriptors can be for character special files, pipes, sockets, or files.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1POL):</b>	31-bit
<b>AMODE (BPX4POL):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts

**Operation****Environment****Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1POL, (PollArrayPtr,
               NMsgsFds,
               Timeout,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4POL with the same parameters. The PollArrayPtr parameter is a doubleword.

**Parameters****PollArrayPtr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains a pointer to an array of Pollfd structures. The elements of the array must be arranged such that the Pollfd structures that contain file descriptors precede the Pollfd structures that contain message queue identifiers, if any are specified.

There is one Pollfd structure for each file descriptor or message queue that is being polled. A Pollfd structure specifies the file descriptor or message queue and the events for which it is being polled. On return, the poll service sets the corresponding bit in the response section of the Pollfd structure if the requested condition is true.

The events that can be polled are:

**POLLRDNORM**

Normal data can be read without blocking.

**POLLRDBAND**

Data from a nonzero priority band can be read without blocking. For STREAMs, this flag is set in revents, even if the message is of zero length.

**POLLIN**

Same as POLLRDNORM.

**POLLWRNORM**

Normal data may be written without blocking.

**POLLWRBAND**

Priority data (priority band greater than 0) may be written.

**POLLPRI**

Out-of-band data may be received without blocking.

**POLLOUT**

Same as POLLWRNORM.

**POLLNVAL**

The specified fd/msgid value is not valid. This flag is only valid in the revents bitmask; it is ignored in the events bitmask.

**POLLERR**

An error has occurred. This flag is only valid in the revents bitmask; it is ignored in the events bitmask.

**POLLHUP**

The device has been disconnected. This event and POLLOUT are mutually exclusive; a stream can never be writable if a hang-up has occurred. However, this event and POLLIN, POLLRDRNORM, POLLRDBAND, or POLLPRI are not mutually exclusive. This flag is valid in the revents bitmask. It is ignored in the events member.

For more information about the format of this field, see [“BPXPOLL — Map poll syscall parameters”](#) on page 1124. The flags in the events bitmask are defined as:

- POLLEPRI, POLLEWRBAND, POLLEWRNORM, POLLEOUT, POLLEIN, POLLERDBAND, POLLERDNORM

The flags in the revents bitmask are defined as:

- POLLRINVAL, POLLRHUP, POLLRERR, POLLRPRI, POLLRWRBAND, POLLRWRNORM, POLLROUT, POLLRIN, POLLRRDBAND, POLLRRDNORM

**NMsgsFds**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains two numbers, the sum of which gives the total number of PollFd structures pointed to by PollArrayPtr.

The first number, which is in the first halfword of the fullword, tells how many message queue PollFd structures were specified. This number must not exceed 32,767. The second number, which is in the second halfword of the fullword, tells how many file descriptor PollFd structures were specified. This number should not exceed 65,535.

**Timeout**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains a timeout value, in milliseconds, that controls how the file descriptors/message queues are checked.

1. No waiting:

If the Timeout value is 0, poll returns immediately after checking the selected descriptors and queues; no waiting is done.

2. Wait for a specified period of time:

If the Timeout value is greater than 0, it specifies the number of milliseconds to wait for one of the events to occur before returning to the caller. (1000 milliseconds equal 1 second).

3. Wait forever.

If the timeout value is -1, poll blocks until a requested event occurs or until the call is interrupted.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the poll service returns one of the following:

- The number of events that were found to be ready.

The return\_value is similar to NMsgsFds. The first halfword of return\_value contains the number of message queues with ready events. The second halfword contains the number of file descriptors with ready events.

- 0, if the timeout value expired before any of the events were met.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the poll service stores the return code. The poll service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The poll service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	The allocation of internal data structures failed, but a subsequent request may succeed.
<b>EINTR</b>	The select service request was interrupted by a signal for the caller.
<b>EINVAL</b>	One of the parameters specified a value that was not correct. Consult the reason code to determine the exact reason for the error. The following reason codes can accompany this return code: JRWaitForever, JRInvalidNfds, JRNoFdsTooManyQIds.
<b>EIO</b>	One of the descriptors in the poll mask has become inoperative and it is being repeatedly included in a poll, even though other operations against this descriptor have been failing with EIO. A socket descriptor can become inoperative, for example, if TCP/IP is shut down. When a descriptor fails, a failure from poll cannot tell you which descriptor has failed, so generally poll will succeed, and these descriptors will be reported to you as being ready for whatever events were specified on the poll. When the inoperative descriptor is subsequently used on a receive or other operation, you will receive the EIO failure, and can then react to the problem with the individual descriptor. In general, you would close() the descriptor and remove it from the next poll mask. If the individual descriptor's failing return code is ignored, though, and an inoperative descriptor is repeatedly polled and used (even though each time it is used the call fails with EIO), eventually the poll call itself will fail with EIO.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the poll service stores the reason code. The poll service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. Poll bits are supported as follows:

### Regular Files

Always poll true for reading and writing. This means that all poll read and write bits are supported. They will never return with POLLERR or POLLHUP.

### FIFOs / PIPEs

Do not have the concept of out-of-band data or priority band data. They support POLLIN, POLLRDNORM, POLLOUT, POLLWRNORM, and POLLHUP. They ignore POLLPRI, POLLRDBAND, and POLLWRBAND. They never return POLLERR.

### TTYs / OCS

Same support as FIFOs and PIPEs, except that TTYs can return POLLERR.

### Sockets

Have the concept of out-of-band data. They support POLLIN, POLLRDNORM, POLLOUT, POLLWRNORM, and POLLPRI for out-of-band data. They ignore POLLRDBAND and POLLWRBAND. They might return POLLERR, but they will never return POLLHUP.

2. If the value of fd/msgid is less than 0, events is ignored and revents is set to 0 in that entry on return from poll.
3. In each pollfd structure, poll clears the revents member, except that where the application requested a report on a condition by setting one of the bits of events listed previously, the poll service sets the corresponding bit in revents if the requested condition is true. In addition, poll sets the POLLERR flag in revents if the condition is true, even if the application did not set the corresponding bit in events.
4. The poll request is not affected by the O\_NONBLOCK flag.
5. A file descriptor for a socket that is listening for connections indicates that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously indicates that it is ready for writing, once a connection has been established.
6. You can use the poll service as a timer-only function by specifying a zero for PollArrayPtr, NMsgsFds, and by specifying a Timeout value. If you specify zero for Timeout, the poll immediately returns to the caller.

## Characteristics and restrictions

None.

### Examples

See [“BPX1POL \(poll\) example”](#) on page 1270.

## prctl (BPX1PRC, BPX4PRC) — Managing processes and threads

### Function

The **prctl** service sets or queries the attributes of various processes or threads, depending on the subfunction that was requested.

For the macro, see [“BPXYCONS — Constants used by services”](#) on page 1062.

The subfunctions are summarized in the following table.

Subfunction name	Function
PR_GET_NAME	Returns the thread name. See also “ <a href="#">pthread_tag_np (BPX1PTT, BPX4PTT)</a> – Set, query, or both set and query the caller's thread tag data ” on page 615. This function uses Arg2Char.
PR_SET_NAME	Sets the thread name. See also “ <a href="#">pthread_tag_np (BPX1PTT, BPX4PTT)</a> – Set, query, or both set and query the caller's thread tag data ” on page 615. This function uses Arg2Char.
PR_GET_NO_NEW_PRIV	Returns the current value of the no new privileges attribute.
PR_SET_NO_NEW_PRIV	Sets the no new privileges attribute for the process.
PR_GET_DUMPABLE	Returns the current state of the dumpable attribute, which controls whether core dumps can be taken.
PR_SET_DUMPABLE	Sets the state of the dumpable attribute, which determines whether core dumps are produced for the calling process after the delivery of a signal whose default behavior is to produce a core dump.
PR_GET_PDEATHSIG	Returns the signal that this process receives when this process terminates, if the parent process was terminated first.
PR_SET_PDEATHSIG	Sets the signal that this process receives when this process terminates, if the parent process was terminated first.
PR_GET_CHILD_SUBREAPER	Returns the process to receive notice when this process terminates, if the parent process has terminated first.
PR_SET_CHILD_SUBREAPER	Sets the process to receive notice when this process terminates, if the parent process has terminated first.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PRC)</b>	31-bit
<b>AMODE (BPX4PRC)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1PRC,(Subfunction,
               Arg2Int,
               Arg2Char,
               Arg3,
               Arg4,
               Arg5,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PRC with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## **Parameters**

### **Subfunction**

Supplied parameter.

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword that contains one of the sub function constants PR\_XXX.

### **Arg2Int**

Supplied parameter.

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword that contains an integer arg2 input/output parameter for subfunctions other than PR\_GET\_NAME and PR\_SET\_NAME. Must be passed as an input parameter for these set functions:

PR\_SET\_NO\_NEW\_PRIVS

PR\_SET\_DUMPABLE

PR\_SET\_PDEATHSIG

PR\_SET\_CHILD\_SUBREAPER

Output parameter for PR\_GET\_CHILD\_SUBREAPER and PR\_GET\_PDEATHSIG. Must be specified for PR\_GET\_NO\_NEW\_PRIVS and PR\_GET\_DUMPABLE but the value is returned for those functions in the RETURN\_VALUE parameter instead of Arg2Int.

Must be specified for functions PR\_GET\_NAME and PR\_SET\_NAME in which case it will be returned unchanged.

### **Arg2Char**

The name of a char(16) area that is used as an input/output parameter for subfunctions PR\_GET\_NAME and PR\_SET\_NAME. Must be specified for other functions in which case it will be returned unchanged.

#### **Type**

Character

#### **Length**

16

### **Arg3, Arg4, Arg5**

Supplied parameters.

#### **Type**

Integer

#### **Length**

Fullword.

### **Return\_code**

Returned parameter.

#### **Type**

Integer



**Length**

Fullword

The name of a fullword in which the prctl service stores the return code. The prctl service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The prctl service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	An invalid value was specified in <i>flags</i> .
<b>EMFILE</b>	The user limit on the total number of inotify() instances was reached.
<b>EMFILE</b>	The per-process limit on the number of open file descriptors was reached.
<b>ENFILE</b>	The system-wide limit on the total number of open files was reached.
<b>ENOMEM</b>	Not enough kernel memory is available.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the prctl service returns 0 or the output of the PR\_GET\_DUMPABLE or PR\_GET\_NO\_NEW\_PRIVS function if the request is successful, or -1 if it is not successful. See the descriptions of Arg2Int and Arg2Char for return values of functions other than PR\_GET\_DUMPABLE and PR\_GET\_NO\_NEW\_PRIVS.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

A fullword in which the **prctl** service stores the reason code. The **prctl** service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value.

**Usage notes**

There are no usage notes for the **prctl** service.

**Related services**

There are no related services for the **prctl** service.

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PRC \(prctl\) example” on page 1271](#) and [“BPX4PRC \(prctl\) example” on page 1363](#).

## pread() and pwrite() (BPX1RW, BPX4RW) – Read from or write to a file without changing the file pointer

---

### Function

The pread() and pwrite() callable service reads from or writes to a given position in a file without changing the file pointer.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1RW):</b>	31-bit
<b>AMODE (BPX4RW):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1RW,(File_descriptor,
              Fuio_Address,
              Fuio_Alet,
              Fuio_Length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RW with the same parameters. The Fuio\_Address parameter is a doubleword.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor of an open file.

#### Fuio\_Address

Supplied parameter

##### Type:

Address

##### Length:

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the Fuio control block, which contains the user request. This area is mapped by the BPXYFUIO macro (see [“BPXYFUIO — Map file system user I/O block”](#) on page 1078).

The setting of the FuioAddr64 bit, and not the AMODE of the caller, indicates whether the buffer address is a 31-bit or 64-bit address. If FuioAddr64 is on, the buffer address is in FuioBufferAddr, and is 31-bit. If FuioAddr64 is off, the buffer address is in FuioBuffV64Addr, and is 64-bit.

#### **Fuio\_Alet**

Supplied parameter

##### **Type:**

Address

##### **Length:**

Fullword

The name of a fullword field that contains the address of Fuio\_alet.

#### **Return\_value**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the pread() and pwrite() service returns the number of bytes that were actually read or written, if the request is successful; or -1, if it is not successful.

#### **Return\_code**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the pread() and pwrite() service stores the return code. The pread() and pwrite() service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. In addition to the return codes listed for the read and write callable services, the pread() and pwrite() service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	The offset argument is not valid. The value is negative.
<b>ENXIO</b>	A request was outside the capabilities of the device.
<b>EOVERFLOW</b>	The file is a regular file and an attempt was made to read or write at or beyond the offset maximum associated with the file.
<b>ESPIPE</b>	File_descriptor is associated with a pipe or FIFO.

#### **Reason\_code**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the pread() and pwrite() service stores the reason code. The pread() and pwrite() service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies

the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Related services

- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket”](#) on page 652
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket”](#) on page 1032

## Characteristics and restrictions

BPX1RW/BPX4RW does not support conversion using Unicode Services.

## Examples

See [“BPX1RW \(Pwrite\) example”](#) on page 1283.

# prlimit (BPX1PRL, BPX4PRL) — Get and set resource limits for specified processes

---

## Function

The prlimit callable service gets and sets resource limits for the specified process. A *resource limit* is a pair of values. One value specifies the current (soft) limit and the other one specifies the maximum (hard) limit.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PRL)</b>	31-bit
<b>AMODE (BPX4PRL)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PRL,(Process_id
               Resource,
               New_limit,
               Old_limit,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PRL with the same parameters.

## Parameters

### Process\_id

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword whose value specifies the process identifier for the process whose resource limits are to be set or retrieved.

- If Process\_ID is equal to 0, the resource limits are set or retrieved for the calling process.

### Resource

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains a value that indicates the resource for which to get and set the hard and soft limits. This parameter can specify one of the resources that are listed in the following table:

<i>Table 19. Resources that can be limited by prlimit</i>		
Resource	Description	Action when soft limit is exceeded
<b>RLIMIT_MEMLIMIT</b>	Maximum amount of usable storage above the bar (in 1-MB segments) that can be allocated at one time.	Any attempt to use more above the bar storage than is specified by the limit will fail.
<b>RLIMIT_CORE</b>	Maximum core file size (in bytes) created by a process. A value of 0 prevents core file creation.	Core file writing ends at this size.
<b>RLIMIT_CPU</b>	Maximum amount of CPU time (in seconds) used by a process.	SIGXCPU is sent to the process, and the process is granted a small extension to allow for signal generation and delivery. If the extension is used up, the process is stopped with a SIGKILL.
<b>RLIMIT_FSIZE</b>	Maximum file size (in bytes) created by a process. A value of 0 prevents the creation of new files and the expansion of existing files.	SIGXFSZ is sent to the process. If the process is blocking, catching, or ignoring SIGXFSZ, continued attempts to increase the size of a file beyond the limit fail with a return code of EFBIG.
<b>RLIMIT_NOFILE</b>	Maximum number of open file descriptors for a process. This number is one greater than the maximum value that can be assigned to a newly created descriptor.	Functions that create new file descriptors after the limit is reached fail with a return code of EMFILE.

Table 19. Resources that can be limited by prlimit (continued)		
Resource	Description	Action when soft limit is exceeded
<b>RLIMIT_AS</b>	Maximum address space size (in bytes) for a process.	The mmap and shmat callable services fail with a return code of ENOMEM. User getmain and storage obtain requests fail (for example, runtime library stack and heap expansion fails).

**New\_Limit**

Supplied.

**Type**

Structure

**Length**

The length of the rlimit structure.

The name of the rlimit structure that contains the new values for the hard (maximum) and soft (current) limits for the resource that is identified by resource parameter. The BPXYRLIM macro defines the Rlimit structure. (See [“BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142.](#)) Each limit value contains two fullwords. For all resources except RLIMIT\_FSIZE and RLIMIT\_MEMLIMIT, the upper fullword for each limit value is ignored.

**Old\_Limit**

Returned.

**Type**

Structure

**Length**

The length of the rlimit structure.

The name of the rlimit structure in which the hard (maximum) and soft (current) limits for the resource that is identified by the resource parameter are to be placed. The BPXYRLIM macro defines the rlimit structure. (See [“BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142.](#)) Each limit value contains two fullwords. For all resources except RLIMIT\_FSIZE and RLIMIT\_MEMLIMIT, the upper fullword for each limit value is ignored.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the prlimit service returns a value of zero if it is successful and -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the prlimit service stores the return code. The prlimit service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The prlimit service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	<p>One of the input parameters was not valid.</p> <ul style="list-style-type: none"> <li>• The resource that was specified is not valid.</li> <li>• The soft limit that is to be set exceeds the hard limit to set.</li> <li>• The soft limit that is to be set is below the current usage.</li> <li>• The hard limit that is to be set exceeds a system-defined limit.</li> <li>• One of the file size limits that was specified is a negative value.</li> </ul> <p>The following reason codes can accompany the return code: JrInvalidResource, JrSoftExceedsHard, JrSoftBelowUsage, JrFdOpenAboveLimit, JrOpenFileLimitMax, JrNegFileSizeLimit, or JrOutOfRange.</p>
<b>EMVSAF2ERR</b>	<p>A security product internal error occurred. Consult the Reason_code parameter for the exact reason for the error.</p>
<b>EPERM</b>	<p>The calling process does not have appropriate privileges.</p> <ul style="list-style-type: none"> <li>• An attempt was made to raise a hard (maximum) limit, but the calling process did not have superuser authority.</li> <li>• The calling process is not permitted to get or set limits for the process that is identified by the pid parameter. The caller must have superuser authority or have the real, effective, and saved set user IDs of the target process match the caller's real user ID and the real, effective, and saved set group IDs must match the caller's real group ID.</li> </ul> <p>The following reason code can accompany the return code: JrRaiseHardLimit or JrNotpermittedToPid.</p>
<b>ESRCH</b>	<p>There is no process with a process ID equal to Process_ID.</p> <p>The following reason code can accompany the return code: JrInvalidPid.</p>
<b>Reason_code</b>	Returned parameter.
<b>Type</b>	Integer
<b>Length</b>	Fullword
<p>The name of a fullword in which the prlimit service stores the reason code. The prlimit service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For a list of reason codes, see <a href="#">Reason codes</a> in <i>z/OS UNIX System Services Messages and Codes</i>.</p>	

## Usage notes

1. You can modify the soft limit to any value that is less than or equal to the hard limit. For the RLIMIT\_CPU, RLIMIT\_NOFILE, and RLIMIT\_AS resources, if prlimit is called with a new soft limit that is lower than the current usage, prlimit fails with an EINVAL return code. An exception to this rule occurs when the process is running in an address space that contains multiple processes. When you change the RLIMIT\_CPU, you can set a new soft limit that is greater than the time limit of the current process, yet greater than the time consumed by the address space. The prlimit call will succeed and a SIGXCPU signal is generated. The alternative is not to run multiple processes in the same address space.
2. The hard limit can be lowered to any value that is greater than or equal to the soft limit.
3. The hard limit can only be raised by a process that has superuser authority.

4. The caller can retrieve or set the resource limits of another process only when the caller either has superuser authority or the real, effective, and saved set users IDs of the target process match the callers real user ID and the real, effective, and saved set group IDs match the callers real group ID. If not permitted, prlimit returns an EPERM return code.
5. Both Old\_Rlimit and New\_Rlimit can be specified on a single call. If both old and new limits, the hard and soft limits at the time of the service, are the values that are placed into Old\_Rlimit (as opposed to the new values that were just set as a result of the service).
6. If neither Old\_Rlimit and New\_Rlimit structures are input, the function is treated as a no-op.
7. Both the soft limit and the hard limit can be changed with a single call to the prlimit service.
8. If the prlimit service is called with a soft limit that is greater than the hard limit, prlimit returns an EINVAL return code.
9. The resource limit values are propagated across the exec, fork, and spawn services. An exception exists for the exec and spawn for services. If a daemon process invokes the exec service and it has previously invoked setuid, or invokes the spawn service specifying that a user ID change is to occur, then the limit values are set based on the values that are found in the OMVS segment of the target user ID. If the target user's OMVS segment does not specify limit values, then the limit values are set based on the values that are found in the BPXPRMxx parmlib member.
10. For processes in a multiprocess address space, the RLIMIT\_CPU and RLIMIT\_AS limits are shared with all the processes within the address space. For RLIMIT\_CPU, when the soft limit is exceeded, action is taken on the first process within the address space. If the action is termination, all the processes within the address space are terminated.
11. In addition to the RLIMIT\_CORE limit values, CORE dump defaults are set by SYSMDUMP defaults. See [IEADMR00 \(ABDUMP written to a SYSMDUMP data set\)](#) in *z/OS MVS Initialization and Tuning Reference* information on setting up SYSMDUMP defaults via the IEADMR00 parmlib member.
12. Core dumps are taken in 4160-byte increments. Therefore, RLIMIT\_CORE values affect the size of core dumps in 4160-byte increments. For example, if the RLIMIT\_CORE soft limit value is 0, no core dumps are taken. If the RLIMIT\_CORE soft limit value is 8000, the maximum size of a core dump is 8000 \* 4160 bytes.
13. Limits can have an infinite value of RLIM\_INFINITY. MEMLIMIT cannot exceed 16383G, which is defined RLIM\_MEGINFINITY.
14. The hard limit for RLIMIT\_NOFILE cannot exceed the system-defined limit of 524287. A value of RLIM\_INFINITY for RLIMIT\_NOFILE indicates that the current system maximum value should be set.
15. The soft limit for RLIMIT\_NOFILE must be set higher than the value of the highest open file descriptor. Attempting to lower the soft limit to a value that is less than or equal to the highest open file descriptor results in an EINVAL return code.
16. Setting a limit of 0 for RLIMIT\_FSIZE prevents the creation of new files and the expansion of existing files.
17. When RLIM\_INFINITY (X'7FFFFFFF') is passed on a prlimit request, no limit is enforced by prlimit. As a result, the maximum allowable limit is set, regardless of the resource. The RLIM\_MEMLIMIT service treats RLIM\_INFINITY as a request for 21474836471 1-megabyte pages.
18. When the MEMLIMIT is set by z/OS UNIX, the highest value that is supported is 16383 petabytes, or X'FFFC000000000000'.
19. Processes can use this service to control CPU resource consumption. For more information, see [BPXPRMxx \(z/OS UNIX System Services parameters\)](#) in *z/OS MVS Initialization and Tuning Reference*.

## **Related services**

- [“getrlimit \(BPX1GRL, BPX4GRL\) — Get resource limits ” on page 313](#)
- [“getrusage \(BPX1GRU, BPX4GRU\) — Get resource usage ” on page 315](#)
- [“setrlimit \(BPX1SRL, BPX4SRL\) — Set resource limits ” on page 789](#)



**Examples**

See “BPX1PRL (prlimit) example” on page 1271 and “BPX4PRL (prlimit) example” on page 1363.

## pthread\_cancel (BPX1PTB, BPX4PTB) – Cancel a thread

---

**Function**

The pthread\_cancel callable service generates a cancellation request for the target thread.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PTB)</b>	31-bit
<b>AMODE (BPX4PTB)</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1PTB, (Thread_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PTB with the same parameters.

**Parameters****Thread\_ID**

Supplied parameter

**Type:**

Character string

**Length:**

8 bytes

The name of an 8-byte field that contains the thread ID for the thread that is to be canceled.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_cancel service returns 0 if the thread is canceled or the cancel is pending, or -1 if a failure occurs.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_cancel service stores the return code. The pthread\_cancel service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_cancel service can return one of the following values in the Return\_code parameter:

Return Code	Explanation
<b>EINVAL</b>	The value that was specified by thread ID is not valid. It does not contain a value that is consistent with thread IDs managed by the system. The following reason code can accompany this return code: JRLightWeightThID.
<b>ESRCH</b>	The value that was specified by Thread_ID does not refer to a thread that currently exists. The following reason codes can accompany this return code: JRThreadNotFound and JRAlreadyTerminated.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_cancel service stores the reason code. The pthread\_cancel service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. A successful call to pthread\_cancel generates a cancellation request for the target thread.
2. Delivery of the cancelation request either causes a nonretryable 422 abend (with reason code 01A0), or causes the signal interface routine (established with BPX1MSS/BPX4MSS) to receive control.
3. If the invoking process sets \_BPXK\_FORCE\_CANCEL=YES, this service is allowed to cancel threads that are not cancellable when this environment variable is set to NO, which is the default. To do this, the pthread\_cancel() service will wait up to three seconds for the thread cancellation to take effect before terminating the target task with a 422 non-retryable abend, ReasonCode=1A0. The abend occurs only if after three seconds the thread has not terminated. If the target of the pthread\_cancel() is the invoking thread, the service exits without waiting three seconds and the cancellation occurs upon exit from the pthread\_cancel() service. If the invoking process sets \_BPXK\_FORCE\_CANCEL=YES and then cancels a large number of threads, the amount of time to complete the cancels may be significantly larger than when the environment variable is NO. This is because the pthread\_cancel() service may wait for up to three seconds before terminating each thread.
4. See the usage notes in [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state”](#) on page 610 for the definition of thread cancellation points.

## Related services

- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread” on page 581](#)
- [“pthread\\_exit\\_and\\_get \(BPX1PTX, BPX4PTX\) — Exit and get a new thread ” on page 588](#)
- [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread” on page 592](#)
- [“pthread\\_kill \(BPX1PTK, BPX4PTK\) — Send a signal to a thread” on page 595](#)
- [“pthread\\_self \(BPX1PTS, BPX4PTS\) — Query the thread ID ” on page 609](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1PTB \(pthread\\_cancel\) example ” on page 1272.](#)

# pthread\_create (BPX1PTC, BPX4PTC) — Create a thread

---

## Function

The pthread\_create callable service creates new threads in the calling process. Each thread that is created represents a single flow of control within the process with its own unique attributes.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PTC):</b>	31-bit
<b>AMODE (BPX4PTC):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTC,(Init_rtn_addr,
              Work_area_addr,
              Attribute_area_addr,
              Thread_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTC with the same parameters. Init\_rtn\_addr, Work\_area\_addr and Attribute\_area\_addr are doublewords.

**Parameters****Init\_rtn\_addr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the initialization routine for the thread that is to be created. This routine is given first control when a new thread task is created to run the thread. In both AMODE 31 and AMODE 64, the actual address of the initialization routine is a 31-bit address.

**Work\_area\_addr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a user-supplied work area that is later passed to the initialization routine. This address is in the parameter list that is returned by pthread\_exit\_and\_get on a thread get request. For a description of this parameter list, see [“BPXYPTXL — Map the parameter list for pthread\\_create” on page 1141.](#)

**Attribute\_area\_addr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the pthread attribute area that is used to define the attributes of the thread to be created. If a zero address is specified, the attributes are set to their default value. For the mapping of the pthread attribute area and the definition and defaults of the supported attributes, see [“BPXYPTAT — Map attributes for pthread\\_exit\\_and\\_get” on page 1127.](#) The address of the pthread attribute area is in the parameter list that is returned by pthread\_exit\_and\_get on a thread get request. The BPXYPTXL macro also has a description of this parameter list; see [“BPXYPTXL — Map the parameter list for pthread\\_create” on page 1141.](#)

**Thread\_ID**

Returned parameter

**Type:**

Character string

**Length:**

8 bytes

The name of an 8-byte field in which the service returns the thread ID for the thread that is created. This field is valid only if the service returns successfully with a return value of 0.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_create service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_Code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the pthread\_create service stores the return code. The pthread\_create service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_create service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EINVAL</b>	One of the parameters contains a value that is not correct. Consult Reason_code to determine the exact reason that the error occurred. The following reason codes can accompany this return code: JRPtatEye, JRPtatSysLen, JRPtatSysOff, JRPtatLen, JRInitRtn, JRShSpMask, JRPtatWeight, JRPtatDetachState, and JRPtatSyncType.
<b>EAGAIN</b>	The system lacked the necessary resources to create the new thread.
<b>EINVAL</b>	The pthread_create service was requested in a multiprocess/multiuser process. The following reason code can accompany this return code: JRMultiProcUser.
<b>ENOMEM (132)</b>	Not enough space is available.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the pthread\_create service stores the reason code. The pthread\_create service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

The thread initialization routine:

1. The pthread-creating task initialization routine has a user-specified routine to initialize the user environment for each new task that is created to process thread requests, and to control the processing of each thread that is to be run on that task.
2. The pthread-creating task initialization routine is first given control when a new MVS task is created to process a thread request. At this point, the initialization routine should set up the user environment for the new task. After performing its initialization, the initialization routine can retrieve the first thread to process by invoking the pthread\_exit\_and\_get callable service.
3. This routine performs its own initialization and cleanup processing for each thread that is to be processed.
4. When this routine gains control, signals and cancellation requests are blocked.
5. The environment in which the initialization routine receives control is described in the following table:

Operation	Environment
<b>Authorization:</b>	Problem program, key that is inherited from TCB key of initial pthread creating task
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE:</b>	31(64), the same as the caller of pthread_create
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Serialization:</b>	Enabled for interrupts
<b>Locks:</b>	No locks held
<b>Control parameters:</b>	All parameters addressable in Primary

6. Upon entry to the initialization routine, the register contents are as follows:

- R1 contains the address of a standard MVS parameter list. In AMODE 31 the parameter list consists of two 4-byte pointers. In AMODE 64 the parameter list consists of two 8-byte pointers. The parameter list consists of the following parameters:
  - a. The address of an initial work area for use by the initialization routine during its setup processing.
  - b. The address of a fullword field that contains the length of the initial work area.
- R2-R12 are unspecified.
- R13 contains the address of a 208-byte save area for use by the initialization routine to perform standard save area linkage and save the general and access registers.
- R14 contains the return address for the initialization routine to return control to the system. This address must be preserved by the initialization routine. When the initialization routine is given control in AMODE 31, the high-order bit (bit 0) of this address is ON. When it is given control in AMODE 64, bits 32 and 33 of the 64-bit R14 are OFF. The initialization routine can always do a simple branch to return to its caller.
- R15 contains the address of the initialization routine.

7. After the first thread request is received, in order for the initialization routine to process subsequent thread requests, it invokes pthread\_exit\_and\_get within a loop. It can then exit the previous thread and obtain a new thread to process.

8. To provide the most efficient interface with the high-level-language environment, the following characteristics apply to the thread initialization routine:

- a. Only one pthread-creating task initialization routine is allowed per process image. When a process image is cleaned up after an invocation of the exec or execmvs service, the address can be changed. If the specified address is different within a given process image, the pthread\_create invocation fails with a return value of -1, a return code of EINVAL, and a reason code of JRInitRtn.
- b. Only one shared subpool mask is allowed per process image. When a process image is cleaned up after an invocation of the exec or execmvs service, the subpool mask can be changed. If the specified shared subpool mask is different within a given process image, the pthread\_create invocation fails with a return value of -1, a return code of EINVAL, and a reason code of JRShSpMask.
- c. The work area and pthread attribute area are passed through from pthread\_create to the caller of pthread\_exit\_and\_get without each being copied. The caller of pthread\_create must therefore ensure that the storage that is provided for these items is not released or modified before these items are used by the caller of pthread\_exit\_and\_get.

9. A minimum of 256 KB is required in the high private below the line. If this is not available, the pthread\_create() terminates with a RC=ENOMEM and a RSNcode=0B510292.

Handling MVS tasks and threads:

Each thread that is created with `pthread_create` runs as an MVS subtask of the initial pthread-creating task (IPT). The IPT is the task that issued the first `pthread_create` call within the address space.

**Note:** The IPT is not the same as the pthread-creating task initialization routine. The IPT refers to the task that the first thread runs on, whereas the pthread-creating task initialization routine is the routine given control when a `pthread_create` is done.

When all the threads created with `pthread_create` and the IPT have ended, the next task in the address space to issue a `pthread_create` call is made the IPT.

Threads that are created by `pthread_create` are represented by 8-character thread IDs. A thread ID is unique only for a given process; multiple processes can have threads that are represented by the same thread ID.

When exiting from the initial pthread-creating task (IPT):

When exiting back to the operating system from the IPT, the caller may receive an A03 abend if any `pthread_created` tasks are still running. These tasks may still be running even if the IPT has called `pthread_join` for all the threads that it created. To avoid the A03 abend, the IPT should call the `_exit` service when it is ready to return to the operating system. The `_exit` service ends the IPT and all of its `pthread_created` subtasks without causing an A03 abend to occur.

Other usage notes

1. The pthread attribute area is passed as input to the `pthread_create` callable service to describe the attributes of the thread that is to be created. The area is split into two sections. The first section is the system attribute area, which is used by the system to build the new thread. The second section is the user area, which is intended for use by the pthread-creating task initialization routine that receives the address of the entire pthread attribute area from `pthread_exit_and_get`.
2. The system offset and user offset fields indicate where the start of each area begins. The system offset field (PTATSYSOFFSET) must be set to (PTATSYSOFFVAL), or `pthread_create` fails with a -1 return value, a return code of EINVAL, and a reason code that indicates the exact error. The user offset field PTATUSEROFFSET must be set to 0 if no user attributes are specified.
3. The system length and user length fields indicate the length of each area. The system length field (PTATSYSENGTH) must be set to PTATSYSENVAL. If it is not, `pthread_create` fails with a -1 return value, a return code of EINVAL, and a reason code that indicates the exact error. The user length field PTATUSERLENGTH can be set to any length. However, if the sum of PTATUSERLENGTH + PTATSYSENGTH does not equal PTATLENGTH, `pthread_create` fails with a -1 return value, a return code of EINVAL, and a reason code that indicates the exact error.
4. The characteristics of each thread attribute and its impact on the `pthread_create` are as follows:
  - **Detach** state specifies the detach state of the thread that is to be created. A thread that is created in a DETACHED state cannot be joined (with the `pthread_join` callable service) by other threads, and has its system-obtained storage freed when it exits. A thread that is created in an UNDETACHED state can be joined by other threads, and does not have its system-obtained storage freed until it has been detached with `pthread_detach`. If the pthread attribute area is not specified on a `pthread_create` invocation, the default value is UNDETACHED.
  - **Weight** specifies the weight of the thread that is to be created. A thread that is created with the MEDIUMWEIGHT attribute allows the executing task to be reused when the thread exits.

The thread is assumed to clean up all resources that it used. Due to system limits, the medium weight thread can sometimes be terminated. Pthreads in a wait for greater than 30 seconds are ended. If your application requires obtained resources to remain intact even if a medium weight thread is terminated, then consider using the `ipt_affinity()` service to associate the resource with the IPT. If the medium weight thread goes away, the resources are still associated with the IPT. If you use `spawn()` to start your application, you can use environment variable `_BPXK_UNUSEDTASKS` to keep idle medium weight threads in a wait. If you have hundreds of idle medium weight threads, system resources are used, which can degrade performance. For more information about the `_BPXK_UNUSEDTASKS` environment variable, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).

When a heavyweight pthread exits, the associated MVS task can no longer request threads to process. If the pthread attribute area is not specified on a pthread\_create invocation, the default value is HEAVYWEIGHT.

- Sync type specifies the synchronous processing type of the thread to be created. The supported sync types are SYNCHRONOUS and ASYNCHRONOUS. A SYNCHRONOUS thread is one that is created only if the resources are immediately available to create it. An ASYNCHRONOUS thread is one that is queued until resources are available. An EAGAIN return code is received from a pthread\_create invocation for a SYNCHRONOUS thread if the resources are not available. This situation can occur if the thread or task limit was already reached for the calling process. If the task limit was reached, only ASYNCHRONOUS threads can be created. If the thread limit was reached, the service fails regardless of the SYNC TYPE. The thread or task limit is specified by parmlib member BPXPRMxx. If the pthread attribute area is not specified on a pthread\_create invocation, the default value is SYNCHRONOUS.
  - Shared Subpool Mask type specifies the set of subpools that are to be shared between threads. The bit positions of the mask represent the subpool number to be shared. If a bit is on, the subpool is shared. Specify subpools 1-127 by turning on their corresponding bit positions in the mask. Turning on the first bit indicates that subpool 1 is to be shared, and so on, to bit position 127. Bit 128 is the enabling bit; if it is off the subpool mask is ignored and the system default is used. The default shared subpools are 1, 2 and 78. The shared subpools must remain constant within the process image; any variation results in the failure of the pthread\_create service.
5. If the calling thread is in a workload management (WLM) enclave, the newly created thread is joined to the same WLM enclave. After the thread is joined, WLM can manage the calling thread and the newly created thread as one "business unit of work" entity for system accounting and management purposes.

## Related services

- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#)
- [“pthread\\_exit\\_and\\_get \(BPX1PTX, BPX4PTX\) — Exit and get a new thread ” on page 588](#)
- [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread” on page 592](#)
- [“pthread\\_kill \(BPX1PTK, BPX4PTK\) — Send a signal to a thread” on page 595](#)
- [“pthread\\_self \(BPX1PTS, BPX4PTS\) — Query the thread ID ” on page 609](#)

## Characteristics and restrictions

To prevent unauthorized programs from gaining control in an authorized environment, pthread\_create does not allow unauthorized callers if the IPT is running in an authorized key (0–7). Unauthorized callers are problem program state, key 8, and not job-step authorized. This restriction is required because the tasks that are created by pthread\_create inherit the TCB key of the IPT.

To prevent deadlocking tasks within an MVS address space, pthread\_create is supported only from the initial pthread\_create task and from any of its daughter tasks. Invocations of pthread\_create from any other tasks fail with a -1 return value, an EMVSERR return code, and a reason code of JRPTCNotSupp.

## Examples

See [“BPX1PTC \(pthread\\_create\) example” on page 1272](#).

# pthread\_detach (BPX1PTD, BPX4PTD) — Detach a thread

---

## Function

The pthread\_detach callable service detaches a thread in the calling process. When a thread is detached, its system storage can be reclaimed when the thread exits.



## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PTD):</b>	31-bit
<b>AMODE (BPX4PTD):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTD, (Thread_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PTD with the same parameters.

## Parameters

### Thread\_ID

Supplied parameter

#### Type:

Character string

#### Length:

8 bytes

The name of an 8-byte field that contains the thread ID for the thread that is to be detached.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the pthread\_detach service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the pthread\_detach service stores the return code. The pthread\_detach service returns Return\_code only if Return\_value is -1. For a list of return

code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The pthread\_detach service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EINVAL</b>	The value that was specified by thread ID is not valid; it does not contain a value that is consistent with thread IDs managed by the system. The following reason code can accompany this return code: JRLightWeightThid.
<b>ESRCH</b>	The system has detected that the value that was specified by thread ID refers to a thread that is already detached or that cannot be found. The following reason codes can accompany this return code: JRThreadNotFound and JRAlreadyDetached.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_detach service stores the reason code. The pthread\_detach service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread”](#) on page 581
- [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread”](#) on page 592

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PTC \(pthread\\_create\) example”](#) on page 1272.

## pthread\_exit\_and\_get (BPX1PTX, BPX4PTX) — Exit and get a new thread

---

**Function**

The pthread\_exit\_and\_get callable service exits a thread, gets a new thread request to process, or both. To start a new thread request, see [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread”](#) on page 581.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PTX):</b>	31-bit

Operation	Environment
<b>AMODE (BPX4PTX):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTX, (Status_field,
               Options_field,
               Signal_setup_userdata,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PTX with the same parameters. The Status\_field and Signal\_setup\_userdata parameters are doublewords.

## Parameters

### Status\_field

Supplied parameter

#### Type:

Integer

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the status of the exiting thread. This status is available to any other thread that uses the pthread\_join service to wait for the termination of this thread.

### Options\_field

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains one of the following option values:

#### PTEXTITTHREAD

Exit the calling thread. This causes the cleanup of system-related resources for the calling thread.

#### PTGETNEWTTHREAD

Exit the last obtained thread and get the next available thread to process. The first invocation of pthread\_exit\_and\_get from the pthread-creating task initialization routine must specify this option.

#### PTFAILIFLASTTHREAD

Exit the calling thread only if it is not the last thread in the process.

The default option value is PTEXTITTHREAD. The option values are defined in the BPXYCONS macro; see [“BPXYCONS — Constants used by services” on page 1062](#). You can combine options by specifying a plus between them.

**Signal\_setup\_userdata**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains 4 bytes (8 bytes) of user data that is normally supplied on the signal setup service, mvssigsetup. This field is used only when the PTGETNEWTHREAD option is specified. If this field contains a zero address, the signal setup user data is not changed for this thread. This field is ignored when the PTEXITTHREAD option is specified.

**Return\_Value**

Returned parameter

**Type:**

Address

**Length:**

Fullword

The name of a fullword in which the service stores the return value. The return value varies depending on the options specified, as follows:

- PTEXITTHREAD option value specified:

**-1**

The caller asked to exit the calling thread, but the thread could not be exited. For an explanation of the error, see Return\_code and Reason\_code.

**0**

The thread was successfully exited.

- PTGETNEWTHREAD option value specified:

**-1**

The caller asked for a new thread to process, but the thread request could not be satisfied. No new thread requests can be handled by the calling task. For an explanation of the error, see Return\_code and Reason\_code.

**>0**

The address of the parameter list for the new thread request that is to be processed. The parameter list consists of the following:

- The user work area address that was specified on the pthread\_create invocation.
- The user attribute area address that was specified on the pthread\_create invocation.
- The address of an 8-byte field that contains the thread ID of the thread request.
- The address of a 4-byte thread run status field. For the possible status values and their definitions, see [“BPXYPTXL — Map the parameter list for pthread\\_create” on page 1141](#).

- PTFAILIFLASTTHREAD option value specified:

**-1**

The caller asked to exit the calling thread only if it was not the last thread, but the thread could not be exited. For an explanation of the error, see Return\_code and Reason\_code.

**0**

The thread was successfully exited.

This parameter list is mapped by the BPXYPTXL macro; see [“BPXYPTXL — Map the parameter list for pthread\\_create” on page 1141](#). The storage for the list is supplied by the system and should not be modified or freed by the caller of pthread\_exit\_and\_get.

**Return\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_exit\_and\_get service stores the return code. The pthread\_exit\_and\_get service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_exit\_and\_get service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EINVAL</b>	One of the parameters contains a value that is not valid. The following reason codes can accompany the return code: JRInvOption, JRGetFirst, JRHeavyWeight, JRQuiesceInProgress, and JRLastThread.

**Reason\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_exit\_and\_get service stores the reason code. The pthread\_exit\_and\_get service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes for pthread\_exit\_and\_get**

1. The pthread\_exit\_and\_get service provides a highly efficient mechanism for processing medium-weight threads. A medium-weight thread is a unit of work that causes reuse of MVS tasks. If a medium-weight thread exits, the task is still capable of processing another medium-weight thread request. The pthread\_exit\_and\_get service provides pthread\_exit with an option that obtains a new thread for its caller to process.
2. The first invocation of pthread\_exit\_and\_get from the pthread-creating task initialization routine must specify the PTGETNEWTHREAD option. On the first invocation, a thread request is retrieved without the occurrence of a thread exit. All subsequent invocations result in a thread exit, following which the next available thread request is obtained. If the PTGETNEWTHREAD option is not specified on the first pthread\_exit\_and\_get invocation, the service fails with a -1 return value, an EINVAL return code, and a JRGetFirst reason code.
3. Using the PTGETNEWTHREAD option can cause a failure if the process is being quiesced. If this happens, the pthread\_exit\_and\_get service fails with a -1 return value, an EINVAL return code, and a JRQuiesceInProgress reason code. At this point, the caller should perform its own cleanup and return to the operating system to allow the task to terminate.
4. If the PTFAILIFLASTTHREAD option is specified and the pthread\_exit\_and\_get is issued from the last thread, the thread is not exited and a JRLastThread reason code is returned with a -1 return value and an EINVAL return code. Any thread that has never issued a pthread\_create or that was not created with pthread\_create is considered the last thread when the PTFAILIFLASTTHREAD option is used.
5. When pthread\_exit\_and\_get is used to get a new thread request, the signal environment is inherited from the creator of the thread. The signal state for the newly created thread is roughly analogous to that of a newly created process after the fork and exec services have been performed. The one exception is that the new thread inherits the setup state from the creator.

6. A successful invocation of pthread\_exit\_and\_get awakens a thread that is waiting for the exiting thread, through the pthread\_join service. The thread exit status that is specified on the pthread\_exit\_and\_get call is made available to the waiting thread.
7. After pthread\_exit\_and\_get is requested with the PTEXITTHREAD option from a given task, that task can no longer request z/OS UNIX services. An exception is the mvspocclp service (BPX1MPC, BPX4MPC), which can be issued to undub the task. The caller should perform its own cleanup and return to the operating system to allow the task to end.
8. If pthread\_exit\_and\_get fails for any reason (with a return value of -1), the caller should perform cleanup and return to the operating system to allow the task to end.
9. When a thread that specified the PTGETNEWTHREAD option is terminated with pthread\_exit\_and\_get and the maximum allowable task limit is exceeded, a JRMaxTasks reason code is returned.
10. When this service is called from the initial pthread-creating task (IPT), it waits for all threads that were created with pthread\_create to end.
11. For information about the pthread attribute area, see [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread” on page 581](#).
12. If you are going to use this service in a multiple-pthread environment, see [Appendix H, “Using threads with callable services,” on page 1421](#).
13. If the PTEXITTHREAD option is used when a non-pthread\_created thread exits, the status of the exiting thread is saved. This status is available to any other thread that uses the pthread\_join service.
14. Up to 10 medium-weight threads that have exited are allowed. The other tasks are terminated with Return Code EmvsErr and Reason code JrMaxTasks. In addition, these 10 threads waiting is only allowed to wait up to 30 seconds for new work. After 30 seconds, they will be terminated with Return Code EmvsErr and Reason code JrMaxTasks
15. If environment variable \_BPXK\_UNUSEDTASKS is set to KEEP, then up to MaxThreadTasks can remain in a pthread\_exit\_and\_get wait. In addition, there is no time limit on how long they can remain in the wait. The \_BPXK\_UNUSEDTASKS environment variable value is not propagated on spawn() with userid or setuid exec.

## Related services

- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread” on page 581](#)
- [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread” on page 592](#)

## Examples

See [“BPX1PTX \(pthread\\_exit\\_and\\_get\) example ” on page 1275](#).

# pthread\_join (BPX1PTJ, BPX4PTJ) — Wait on a thread

---

## Function

The pthread\_join callable service obtains the termination status for a specific thread. The pthread\_join service waits only if the thread has not ended, is not in a detached state, and is not currently joined by another thread.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN

Operation	Environment
AMODE (BPX1PTJ)	31-bit
AMODE (BPX4PTJ)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1PTJ,(Thread_ID,
               Status_field_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PTJ with the same parameters. The Status\_field\_address parameter is a doubleword.

Parameters

**Thread\_ID**  
Supplied parameter

**Type**  
Character string

**Length**  
8 bytes

The name of an 8-byte field that contains the thread ID for the target thread that is to be waited upon.

**Status\_field\_address**  
Supplied parameter

**Type**  
Address

**Length**  
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of a status field in which to return the exit status of the thread that is specified by the thread ID value. If this field is zero, the thread exit status is not returned.

**Return\_Value**  
Returned parameter

**Type**  
Integer

**Length**  
Fullword

The name of a fullword in which the pthread\_join service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_Code**  
Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_join service stores the return code. The pthread\_join service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_join service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EINTR</b>	The calling process received a signal before the completion of an event that would cause the pthread_join service to return. The service was interrupted by a signal. In this case, the value contained in Status_field_address is undefined.
<b>EINVAL</b>	The value that was specified by thread ID is not valid; it does not contain a value that is consistent with thread IDs managed by the system. The following reason code can accompany this return code: JRLightWeightThread.
<b>ESRCH</b>	The value that was specified by thread ID does not refer to a thread that is undetached. The following reason codes can accompany this return code: JRThreadNotFound, JRAlreadyJoined, and JRAlreadyDetached.
<b>EDEADLK</b>	A deadlock was detected; or the value specified by thread ID refers to the calling thread. The following reason codes can accompany this return code: JRJoinLoop and JRJoinToSelf.

**Reason\_Code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_join service stores the reason code. The pthread\_join service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The pthread\_join service can be called repeatedly for a thread until the thread is detached. However, a thread can be the target of only one pthread\_join at a time.
2. In AMODE 31, the status field pointed to by Status\_Field\_Addr is 4 bytes. In AMODE 64, the status field is 8 bytes. See [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread”](#) on page 581 for further information.
3. When pthread\_join is issued for a non-pthread-created thread that has ended, that thread will only be found if it exited via pthread\_exit\_and\_get.

**Related services**

- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread”](#) on page 581
- [“pthread\\_detach \(BPX1PTD, BPX4PTD\) — Detach a thread”](#) on page 586



## Characteristics and restrictions

None.

## Examples

See “BPX1PTJ (pthread\_join) example” on page 1273.

# pthread\_kill (BPX1PTK, BPX4PTK) – Send a signal to a thread

## Function

The pthread\_kill callable service targets a signal to a particular thread. The service is limited to interthread communication within a process.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PTK)</b>	31-bit
<b>AMODE (BPX4PTK)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTK,(Thread_ID,
              Signal,
              Signal_options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTK with the same parameters.

## Parameters

### Thread\_ID

Supplied parameter

### Type

Character string

### Length

8 bytes

The name of an 8-byte field that contains the target thread that is to receive the signal.

### Signal

Supplied parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword field that contains the signal number that is to be sent to the thread that is indicated by the Thread\_ID parameter. This must be one of the signals that is defined in BPXYSIGH macro, or 0.

If the signal is 0, error checking takes place, but no signal is sent. The pthread\_kill service can be called with a signal value of 0, to verify that the Thread\_ID parameter is correct before the signal is sent.

**Signal\_options**

Supplied parameter

**Type**

Bit

**Length**

Fullword

The name of a fullword field that contains the binary flags that describe how the signal is to be handled by both the kernel and the user-supplied signal interface routine (SIR). The signaling options are passed to the SIR in the signal information control block, which is mapped by BPXYPPSD; see “BPXYPPSD — Map signal delivery data” on page 1124. Signal\_options are mapped as follows:

**First 2 bytes**

User-defined bytes that are delivered with the signal to the SIR in the signal information control block. These bytes are mapped by the BPXYPPSD macro.

**Last 2 bytes**

Flag bits, mapped by PPSDKILOPTS, that are defined as follows:

- First bit - signal to bypass Ptrace processing
- Second bit - reserved
- Third bit - signal code that is specified in the first 2 bytes is set by the application
- Remaining bits - reserved

**Return\_Value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_kill service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_Code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_kill service stores the return code. The pthread\_kill service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_kill service can return one of the following values in the Return\_code parameter:

**Return\_code****EINVAL****Explanation**

One of the following conditions causes this return code:

- The value of Signal is not valid, or is not the number of a supported signal.
- The thread corresponding to Thread\_ID was not found, not valid, or ended.

The following reason codes can accompany the return code: JRInvalidSignal, JRLightWeightThid, JRThreadNotFound, and JRThreadTerm.

**Reason\_Code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_kill service stores the reason code. The pthread\_kill service returns Reason\_code only if Return\_value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The pthread\_kill service provides a mechanism for asynchronously directing a signal to a thread in the calling process. This mechanism could be used, for instance, by one thread to cause the processing of other threads within the process.
2. The pthread\_kill service is the only function that can issue the thread-scoped signals (SIGTHSTOP and SIGTHCONT). The SIGTHSTOP signal stops a specific thread; other threads in the process are not affected. The SIGTHCONT signal can be issued by the pthread\_kill service to resume the stopped thread.

SIGTHSTOP and SIGTHCONT can only be issued to threads within the same process. If all the threads in a process are stopped with SIGTHSTOP, the process is virtually hung. No other threads can send a SIGTHCONT signal to wake them up. The stopped threads must be manually killed.

The SIGTHSTOP and SIGTHCONT signals are noncatchable, nonblockable, and cannot be ignored.

**Related services**

- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PTK \(pthread\\_kill\) example” on page 1274](#).

**MVS-related information**

Delivery of a signal to the signal interface routine occurs only when the PSW key of the caller is equal to the signal delivery key of the process. The signal delivery key is set to the PSW key of the caller of the first callable service that dubbed the process.

## pthread\_quiesce (BPX1PTQ, BPX4PTQ) – Quiesce threads in a process

---

### Function

The pthread\_quiesce callable service performs quiesce or query functions on threads. Depending on the function that is specified, pthread\_quiesce queries the thread environment in the current process, or synchronously quiesces all threads in the current process (except for the calling thread, which returns when all threads have been quiesced).

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PTQ)</b>	31-bit
<b>AMODE (BPX4PTQ)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The format is as follows:

```
CALL BPX1PTQ (Quiesce_type,
              User_data,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTQ with the same parameters. User\_data is a doubleword.

### Parameters

#### Quiesce\_type

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains one of the following values:

#### QUIESCE\_TERM

Terminates all threads (except the invoking thread) that were created with pthread\_create and IPT threads, allowing the signal interface routine to receive control when the quiesce request is delivered.

**QUIESCE\_FORCE**

Terminates all threads (except the invoking thread) that were created with pthread\_create, and IPT threads that do not allow the signal interface routine to receive control when the quiesce request is delivered.

**PTHREAD\_QUERY**

Counts the number of threads that were created with pthread\_create or IPT threads and returns the count in Return\_value.

**QUIESCE\_FREEZE**

Freezes all threads (except the invoking thread) in the process, including threads that were created with pthread\_create, IPT, and MVS dubbed tasks. The signal interface routine is allowed to receive control when the quiesce event is delivered.

**QUIESCE\_UNFREEZE**

Continues execution of all threads (except the invoking thread) in the process that are in a frozen state.

**FREEZE\_THIS\_THREAD**

Places the invoking thread into a frozen state, in response to a QUIESCE\_FREEZE request.

The Quiesce\_type values are defined in the BPXYCONS macro; see [“BPXYCONS — Constants used by services”](#) on page 1062.

**User\_data**

Supplied parameter

**Type**

Character string

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) that is to be passed to the signal interface routine when the quiesce request is delivered.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_quiesce service places the return value. The return value varies depending on the Quiesce\_type:

- **PTHREAD\_QUERY quiesce type specified:**

**-1**

The caller asked to query the number of threads that were created with pthread\_create and IPT threads in the process, but the request could not be completed. For an explanation of the error, see the return code and reason code.

**0**

The calling thread is the initial pthread-creating task (IPT), and no other threads that were created with pthread\_create exist in the current process.

**1**

The calling thread is created with pthread\_create, not the IPT, and no other threads that were created with pthread\_create or IPT threads exist in the current process.

**>1**

The value indicates the number of threads that were created with pthread\_create and IPT threads in the current process.

- **All other quiesce types specified:**

**-1**

The caller asked to quiesce a thread in the current process, but the target threads may not all have been quiesced. For an explanation of the error, see the return code and reason code.

**0**

The target threads in the current process were successfully quiesced.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the pthread\_quiesce service stores the return code. The pthread\_quiesce service returns a Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_quiesce service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value specified for Quiesce_type was incorrect. The following reason code can accompany the return code: JRQuiesceTypeInvalid.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_quiesce service stores the reason code. The pthread\_quiesce service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Requesting pthread\_quiesce with the QUIESCE\_TERM or QUIESCE\_FORCE options delivers a quiesce request to the IPT and all pthread\_created threads in the process. When Quiesce\_type is QUIESCE\_TERM, the request is delivered to each thread by the signal interface routine (SIR) if the process is set up to intercept the quiesce request. If the process is not set up for quiesce request interception, or if Quiesce\_type is QUIESCE\_FORCE, the kernel performs the quiesce request for each thread. For details on how to intercept quiesce requests, see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals”](#) on page 487.
2. The kernel issues 422 abends when performing the termination quiesce request. If the request is intercepted by the user-defined SIR, it should perform whatever cleanup is necessary, and then issue the pthread\_exit\_and\_get service to end the thread.
3. Requesting pthread\_quiesce with the QUIESCE\_TERM or QUIESCE\_FORCE options from a thread that is not the IPT, or that was not created with the pthread\_create service, has no effect on any threads in the process; and pthread\_quiesce returns with a 0 return value.
4. The pthread\_quiesce service should be requested with one of the terminating options before an exit (BPX1EXI, BPX4EXI) to prevent the other threads in the process from receiving an asynchronous abend.
5. When requested with one of the terminating options, the pthread\_quiesce service posts all MVS tasks that are in pthread\_exit\_and\_get (BPX1PTX, BPX4PTX) waiting for more work. The pthread\_exit\_and\_get service returns to the caller with a -1 return value. The caller can then clean up the task-related resources before the normal end (SVC 3) of the task.

6. If the pthread\_quiesce service is invoked when Quiesce\_type is PTHREAD\_QUERY from a thread that was not created with pthread\_create and is not an IPT thread, pthread\_quiesce returns with a 0 return value.
7. The use of QUIESCE\_FREEZE is not limited to the IPT and pthread\_created threads. This option causes a quiesce event to be delivered to every other thread in the process. Upon return from pthread\_quiesce, all threads in the process are no longer executing and are in a "frozen state".
8. If the target thread is intercepting quiesce events (see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)), the signal interface routine gains control and is expected to either issue the queue\_interrupt service (this is not a good time to freeze this thread) or issue the pthread\_quiesce service with the FREEZE\_THIS\_THREAD option. However, since the quiescer is waiting for all threads to be placed into a frozen state, the pthread\_quiesce service should be issued as soon as possible. If the target thread is not intercepting, the kernel places the thread into a frozen state.
9. The FREEZE\_THIS\_THREAD function places the thread into a frozen state only if a freeze request is pending on the calling thread. If a request is not pending, the FREEZE\_THIS\_THREAD function does not suspend execution. Control is immediately returned to the caller with a return code of zero.
10. When you want to restart the process, use the pthread\_quiesce service with the QUIESCE\_UNFREEZE option. All threads that are found to be in a frozen state are restarted.
11. The QUIESCE\_TERM and QUIESCE\_FORCE Quiesce\_types for pthread\_quiesce() terminates all threads in the caller's process with the exception of the caller's thread. Some of those threads might have created local processes in the caller's address space and attached subtasks when those processes were created. Before terminating threads in the caller's process, z/OS UNIX must first terminate any local processes that were created by those threads to prevent asynchronous abends of the tasks attached for those processes. This processing is called *quiescing local processes*. Due to blind dubbing, it is possible for a seemingly unrelated task (JST only) to be dubbed a thread in a process during the initial dubbing of the address space. This blind-dubbed thread will be included in the threads terminated by pthread\_quiesce(). Any additional processes that it created are also included when quiescing local processes. For more information about blind dubbing, see [Note 7 in “set\\_dub\\_default \(BPX1SDD, BPX4SDD\) — Set the dub default service” on page 752](#).

## Related services

- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)
- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread” on page 581](#)

## Examples

See [“BPX1PTQ \(pthread\\_quiesce\) example” on page 1274](#).

# pthread\_security\_np, pthread\_security\_applid\_np (BPX1TLS, BPX4TLS) — Create or delete thread-level security

---

## Function

This service creates or deletes the thread-level security environment for the caller's thread. The authorization that is required to invoke this service is one of the following:

- Read or update access to the BPX.SERVER resource profile in the FACILITY class.
- Superuser status.

## Requirements

### Operation

### Environment

### Authorization

Supervisor state or problem state, any PSW key

Operation	Environment
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1TLS)	31-bit
AMODE (BPX4TLS)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1TLS,(Function_code,
               Identity_Type,
               Identity_Length,
               Identity,
               Pass_Length,
               Pass,
               Option_Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TLS with the same parameters.

Parameters

Function\_code

Supplied parameter

Type:

Integer

Length:

Fullword

The name of a fullword that specifies a numeric value that identifies the function that is to be performed. The following Function\_code constants are defined by the BPXYCONS macro. See [“BPXYCONS — Constants used by services” on page 1062](#).

Constant	Function
TLS_CREATE_THREAD_SEC#	Creates a thread-level security environment for the caller's thread. If a thread-level security environment exists, it is deleted before the new environment is created.
TLS_DAEMON_THREAD_SEC#	Creates a thread-level security environment for the caller's thread without the need for a password if the caller meets one of the following requirements: <ul style="list-style-type: none"><li>• READ access to the BPX.DAEMON resource in the FACILITY class and is a superuser,</li><li>• UPDATE access to the BPX.DAEMON resource in the FACILITY class.</li></ul>



**Constant****TLS\_DELETE\_THREAD\_SEC#****Function**

Deletes the thread-level security environment for the caller's thread, if one exists. If the security environment was created with the TLS\_TASK\_ACEE# option, only the POSIX security information is deleted; the task-level ACEE is left alone.

**TLS\_TASK\_ACEE#**

Initializes the UNIX (POSIX) security data for a task that has an existing task-level security environment (task-level ACEE). If the UNIX security data exists for the calling task, the existing UNIX security data is deleted, and a new set of UNIX security data is established.

**TLS\_TASK\_ACEE\_USP#**

Takes an existing USP from a task-level ACEE and extracts the UID and GID information. This information is then used to build a complete MVS and POSIX security environment for the caller's thread.

**Identity\_Type**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that specifies a numeric value that identifies the format of the user identity that is provided in the Identity parameter. Constants are defined by the BPXYCONS macro. See [“BPXYCONS — Constants used by services”](#) on page 1062.

**Constant****TLS\_IDENTITY\_USERID#****Identity Format**

The user identity is in the format of a 1-to-8-character user ID.

**TLS\_IDENTITY\_CERT#**

The user identity is in the form of a certificate control block.

**Identity\_Length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Identity. The specified length must be consistent with the allowable Identity types:

- USERID - 1 to 8 characters
- CERTIFICATE - the length of the constant OCRT\_LEN, defined in the BPXYOCRT control block. See [“BPXYOCRT — Map the OE certificate support structure”](#) on page 1111.

**Identity**

Supplied parameter

**Type:**

Character string or number, if using an identity type of USERID; structure, if using an identity type of CERTIFICATE.

**Character set:**

Not applicable for an identity type of CERTIFICATE. For an identity type of USERID, the XPG4 portable character set that includes upper and lowercase letters (A-Z, a-z), numerics (0-9), period

(.), dash (-), and underscore (\_). In addition, the special characters \$, %, and # can be specified. (Since these characters are not part of the XPG4 portable character set, the future possibility of program portability should be considered before those these characters are used.)

**Length:**

Specified by the Identity\_Length parameter

If the identity type is specified as TLS\_IDENTITY\_USERID#, this area is the name of a field that contains the user identity in the specified format.

If the identity type is specified as TLS\_IDENTITY\_CERT#, this area is mapped by the BPXYOCRT macro (see [“BPXYOCRT — Map the OE certificate support structure”](#) on page 1111).

**Pass\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket, or between 9 and 100 characters for a password phrase. A length of zero indicates that the Pass parameter is to be ignored.

**Pass**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pass\_length parameter

The name of a field, of length Pass\_length, that contains, left-justified, the password, PassTicket, or password phrase that is to be verified.

**Option\_Flags**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword binary field that contains the pthread\_security\_np options. If no options are required, specify the name of a fullword field that contains 0.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword where the pthread\_security\_np service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_security\_np service stores the return code. The pthread\_security\_np service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_security\_np service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Permission is denied; the specified password is incorrect. The following reason code can accompany the return code: JROK.
<b>EMVSEXPIRE</b>	The password for the specified identity has expired. The following reason code can accompany the return code: JROK.
<b>EINVAL</b>	<p>One or more of the following conditions were detected:</p> <ul style="list-style-type: none"> <li>• The Function_Code that was specified is undefined.</li> <li>• The Identity_Type that was specified is undefined.</li> <li>• The Identity_Length that was specified was not valid for the Identity_Type.</li> <li>• The Password_Length that was specified was not in the range 0 to 8.</li> <li>• An undefined option flag was set.</li> </ul> <p>The following reason codes can accompany the return code: JRTLSCertIDLenInvalid, JRTLSCertTypeInvalid, JRTLSCertLengthInvalid, JRTLSRequestInvalid, JRTLSIdTypeInvalid, JRTLSIdLengthInvalid, JRTLSAddressLengthInvalid, and JRBadOptions.</p>
<b>EMVSERR</b>	An MVS environmental error has been detected. The following reason codes can accompany the return code: JRTLSCallerIsIPT, JRSecActive, JRTLSNotDoneByOE, JRNoPtraceTaskSec, JRNotWLMACEE, JRUnexpectedError, JRTLSDoneOnIPT, JRNoTaskACEE, JRSAFNoUID, JRSAFNoGID, JRSAFNoUSER, JRSAFGroupNoOMVS, JRSAFUserNoOMVS, JRUnexpectedError, and JRSAFInternal.
<b>EPERM</b>	<p>One or more of the following conditions were detected:</p> <ul style="list-style-type: none"> <li>• The calling address space is not authorized to use this service.</li> <li>• A password was not supplied and the RACF SURROGAT class was not activated; or no SURROGAT class profile was defined for the specified user identity.</li> <li>• A password was not supplied and the caller's address space does not have READ permission to the specified RACF SURROGAT class profile of the specified user identity.</li> <li>• A load from an unauthorized (not Program Control protected) library was done in the address space.</li> </ul> <p>The following reason codes can accompany the return code: JRNotServerAuthorized, JRSurrogateUndefined, JRNoSurrogatePerm, JRNoChangeIdentity, and JREnvDirty.</p>
<b>EMVSSAF2ERR</b>	An error occurred in the security product. Consult Reason_code to determine the exact reason that the error occurred. The following reason codes can accompany the return code: JRCertInvalid, JRRACFBlankExits, JRSAFInternal, and JRSAFParmlistError. The reason code can also contain the RACF return and reason codes, respectively, in the two low-order bytes. For more information, see <a href="#">Table 13 on page 533</a> .

Return_code	Explanation
<b>EMVSSAFEXTRERR</b>	An error occurred when the security product was called. Consult Reason_code to determine the exact reason for the error. For more information, see <a href="#">“RACF reason and return codes for EMVSSAFEXTRERR”</a> on page 606.
<b>ENOSYS</b>	The function is not supported on this system. The following reason code can accompany the return code: JRNoSecurityProduct.
<b>ESRCH</b>	The identity that was specified is not defined to the security product. The following reason code can accompany the return code: JROK and JRNoCertForUser.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_security\_np service stores the reason code. The pthread\_security\_np service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**RACF reason and return codes for EMVSSAFEXTRERR**

Table 20. RACF reason and return codes for EMVSSAFEXTRERR

RACF reason code	RACF return code	Explanation
28	0	The user's access was revoked.
8	10	An attempt was made to delete the server address space ACEE before invoking initACEE to purge all managed ACEEs.
8	1C	The user ID was revoked or user access to the group was revoked.
8	20	The user does not appropriate RACF access to either the SECLABEL, the SERVAUTH profile, or APPL.

**Usage notes**

- The ability to create a task-level security environment (ACEE) using the function codes TLS\_CREATE\_THREAD\_SEC# and TLS\_DAEMON\_THREAD\_SEC# is a privileged operation. An installation has the following two ways of allowing an application to use this service with these two function codes:
  - For the highest level of security, the installation defines the BPX.SERVER resource profile in the FACILITY class. For the application to access this service, it must be given read access to this profile. In addition, all load modules that are executing in the application's address space must be defined to RACF. For more information about setting up security, see [Establishing UNIX security in z/OS UNIX System Services Planning](#).
  - For a lower security arrangement, you can assign the user ID under which the application is run a UID of 0 so that it operates as a superuser.
- When a task-level security environment is established, the other z/OS UNIX services are divided into two categories:

- Services that are used to access data in the file system base the permission checks on the task-level security. Therefore, a function like `open()` will only work if the identity of the user in the task security environment has permission to the file. The `pthread_security_np` is useful for creating a file server.
- Services that tend to be process-oriented continue to base the permission checking on the security identity of the process. Functions like `kill` only work if the process has permission to send the signal to the target process. The IPC functions of shared memory, message queues, and semaphores are also accessed with the security environment of the process.

The permission checks are done on the first call to this service, and a successful result is remembered so that future calls to the service run faster. Therefore, revoking access to the BPX.SERVER profile in the FACILITY class does not stop a running server from continuing to create task-level security environments.

3. If a thread with a task level security environment issues a `spawn` (non-local only), the child process is created with the identity of the thread. If a thread with a task level security environment issues a `fork`, the child process is created with the identity of the process.
4. Access to most MVS resources is based on the security identity of the thread.
5. Mixed case passwords and password phrases are supported when the installed security product (such as RACF) supports mixed case; otherwise, passwords and PassTickets are folded to uppercase. Non-graphic characters are always folded to blanks.

The contents of the password phrase string are passed unchanged to the installed security product.

6. The specification of a Pass value is optional. The following are some examples of situations in which a server would want to create a task-level security environment without a Pass value:
  - Some servers allow access to a system with a user ID known as ANONYMOUS. The ANONYMOUS user ID is defined to the system with access to data available to the general public. It is up to the installation to define and manage an ANONYMOUS user ID so that integrity is not compromised.
  - Some servers are connected to global security servers. If other services are used to authenticate a user, it is not necessary to provide a Pass value to this service. It is up to the application and the installation to define the level of user authentication that is acceptable.
7. Debugging in this environment is only allowed for users with read permission to the BPX.DEBUG resource profile in the FACILITY class.
8. This service cannot be called from the Initial Pthread Task (IPT) with function code `TLS_CREATE_THREAD_SEC#`. The RTL performs certain process-related functions on the IPT that would be adversely affected by a task-level security environment.
9. If the user identity that is specified by the caller was defined as a SURROGAT user ID and no password was specified, the task-level ACEE that is created for the calling thread has the CLIENT feature turned on. When RACF encounters a task-level ACEE with the CLIENT feature turned on, authority checking is done using both the task and process-level ACEEs. Both ACEEs must have permission to be able to access the resource. For more information about how to define a SURROGAT user, see [Defining servers to process users without passwords](#) in *z/OS UNIX System Services Planning*.
10. If the identity type is `TLS_IDENTITY_CERT#`, the user ID is returned to the caller, filled in with the user ID that is associated with the certificate and ended with a null character.
11. If the function code specified is `TLS_TASK_ACEE#`, the values that are specified for the `Identity_Type`, `Identity_Length`, `Identity`, `Pass_Length`, `Pass`, and `Option_Flags` parameters are ignored. Because the user had the authority to create a task-level ACEE and attach it to the TCB, no additional credentials are necessary to redub the thread with the POSIX identity associated with the user ID of the task-level ACEE.
12. For the `TLS_TASK_ACEE#` and `TLS_TASK_ACEE_USP#` function codes to be used successfully, either the caller must be supervisor state and system key (0-7), or the ACEE for the calling task must have been created by WLM.
13. The POSIX identity established by a `TLS_TASK_ACEE#` can be deleted in one of three ways:

- Issue another TLS\_TASK\_ACEE#. This deletes the old thread-level POSIX identity before establishing the new identity. However, this method fails if the previous thread-level identity was not established by a previous TLS\_TASK\_ACEE#.
  - Issue a TLS\_DELETE\_THREAD\_SEC#. This deletes the POSIX thread-level identity, and the thread takes on the POSIX identity of the process.
  - Issue a pthread\_exit(). If the thread is heavyweight, the task terminates. If the thread is mediumweight, only the POSIX identity is cleaned up; the task-level MVS identity remains.
14. The pthread\_security\_applid\_np() function is equivalent to pthread\_security\_np() with the added feature that it allows an application identifier (*applid*) to be supplied. The applid is used to verify the user's authority to access the application. When a PassTicket is specified, the *applid* is also used in conjunction with the USERID to verify the PassTicket.

If an application is not using the pthread\_security\_applid\_np() function but still wants to pass an *applid* to this service, the application can set the applid value in the BPXYTHLI.

- ThliEP\_FunctionCode is set with ThliEP\_ApplSet.
- ThliEP\_ApplidLen is set to the length of the APPLID. If this value is less than 1 or greater than 8, then the ThliEP\_APPLID value is ignored.
- ThliEP\_APPLID is set to the APPLID value.

If there is no applid value passed, the applid value defaults to OMVSAPPL.

15. If the calling task does not have a USP associated with the task-level ACEE, the kernel treats this call as if the TLS\_TASK\_ACEE# function was called. If a USP is present, the kernel initializes the thread security environment with the UIDs and GIDs of the USP (supp groups are not used). Calling the pthread\_security\_np service with TLS\_DELETE\_THREAD\_SEC# will return the thread to its original state. It is up to the caller to delete the ACEE or USP and maintain the task's TCBSENV field.
16. If the caller's IPT task has previously called BPX1TLS to create a thread-level security environment, then calling BPX1TLS with TLS\_TASK\_ACEE# or TLS\_TASK\_ACEE\_USP# function codes from a pthread will fail.
17. If the pthread\_security\_np service returns a Return\_code of EMVSSAF2ERR and the TLS\_TASK\_ACEE\_USP# function code was specified, then the Reason\_code will contain the propagated Return\_code and Reason\_code from the IRRSGE00 service.
18. Using the TLS\_DAEMON\_THREAD\_SEC# function code without specifying a password is similar to using the BPX.SRV.userid surrogate support. The difference is that the installation does not need to set up individual surrogate profiles for each client that wants a thread-level identity in the target server process. The server can create any identity without authentication as long as it has been given READ permission to the BPX.DAEMON resource in the FACILITY class and has superuser privileges. If UPDATE access is granted to the BPX.DAEMON resource, then superuser privileges are not required.
19. When function code TLS\_CREATE\_THREAD\_SEC# is specified without specifying a PASS parameter, a SURROGAT class check is made, ensuring the caller has access to the profile BPX.SRV.userid (where *userid* is the value that is specified on the IDENTITY parameter). If the *userid* portion of the profile name has blanks in it, then the RACROUTE REQUEST=AUTH results in ABEND282 RC5C. The dump is suppressed and the request fails with a return code of EMVSSAF2ERR and reason code of JrRACFBlankExists.
20. Although z/OS UNIX System Services supports password phrases that are 9-100 characters in length, your installation or the installed security product can have additional rules for password phrase lengths. Ask your security administrator or system programmer if any additional rules apply.
21. The pthread\_security\_np() will recognize and use the ICR passed if a previous \_\_passwd() had been made that specified the same ICR. When \_\_passwd() is used in conjunction with pthread\_security\_np(), an ICR can be used to authenticate a user and then create a thread-level security context (ACEE), similar to using a user ID and password. Because an ICR is a one time use entity, the \_\_passwd() and pthread\_security\_np() services must be called from the same thread and the ICR specified on pthread\_security\_np() and the preceding \_\_passwd() must be identical.

**Note:** An ICR is not a user ID and password. It is a reference in the local identity context cache.

22. If environment variable BPXK\_MIN\_PWFOLD=YES is set then non-graphic characters are not changed to blanks before being passed to the security product. This behavior will exist for all threads in the process where the environment variable was set.

## Restrictions

When the IPT (initial pthread task) is also the initial thread (the first thread created in the process), the IPT can issue the call as many times as it wants. If the IPT is not the initial thread, it can only issue the call once. After that, it will fail with reason code JxTLSDoneOnIPT. If the caller's IPT task has previously called BPX1TLS/BPX4TLS with function codes TLS\_TASK\_ACEE# or TLS\_TASK\_ACEE\_USP#, any call from a thread, whether it is a pthread or not, will fail with reason code JxTLSDoneOnIPT.

If you set the task-level security for the IPT, the process may be adversely affected. Applications should consider these possible effects before they call BPX1TLS/BPX4TLS from the IPT with function codes TLS\_TASK\_ACEE# or TLS\_TASK\_ACEE\_USP#.

- Newly attached pthreads might inherit the task-level security, which may not be the intent.
- The RTL will perform process-related functions that are affected by the IPT task-level security.
- pthreads will no longer be able to issue the call.

## Related services

- [“oe\\_env\\_np \(BPX1ENV, BPX4ENV\) — Examine, change, or examine and change an environmental attribute” on page 500](#)
- [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name” on page 289](#)

## Characteristics and restrictions

The pthread\_security\_np service is restricted to users that have the appropriate privileges.

## Examples

See [“BPX1TLS \(pthread\\_security\\_np\) example” on page 1304](#).

# pthread\_self (BPX1PTS, BPX4PTS) — Query the thread ID

---

## Function

The pthread\_self callable service gets the thread ID of the calling thread.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PTS):</b>	31-bit
<b>AMODE (BPX4PTS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation**

**Control parameters:**

**Environment**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1PTS, (Thread_ID)
```

AMODE 64 callers use BPX4PTS with the same parameter.

**Parameters**

**Thread\_ID**

Returned parameter

**Type:**

Character string

**Length:**

8 bytes

The name of an 8-byte field in which the service places the thread ID of the calling thread.

**Usage notes**

1. The caller should request this service only once when it needs the thread ID of the active thread. It should save a copy of the thread ID in its own storage for repeated usage.
2. If this service fails, the calling thread ends abnormally.

**Related services**

- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread” on page 581](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PTS \(pthread\\_self\) example” on page 1274](#).

## **pthread\_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state**

---

**Function**

The pthread\_setintr callable service sets the specified interruptability state of the calling thread and atomically returns the previous interruptability state.

**Requirements**

**Operation**

**Authorization**

**Dispatchable unit mode**

**Environment**

Supervisor state or problem state, any PSW key

Task



<b>Operation</b>	<b>Environment</b>
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE</b>	31-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1PSI,(Interrupt_state,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PSI with the same parameter.

**Parameters**

**Interrupt\_state**  
Supplied parameter.

**Type**  
Structure

**Length**  
Fullword

Specifies the name of a fullword that contains a numeric value that identifies the interrupt state that is to be set. The following constants, which are defined in the BPXYCONS macro, define the valid states (see “BPXYCONS — Constants used by services” on page 1062):

Constant	Description
<b>PTHREAD_INTR_ENABLE#</b>	When interruptability is enabled, new, or pending cancellation requests are acted upon according to the interruptability type set by the pthread_setintrtype service (BPX1PST, BPX4PST).
<b>PTHREAD_INTR_DISABLE#</b>	When interruptability is disabled, cancellation requests against the target thread are held pending.

**Return\_value**  
Returned parameter.

**Type**  
Integer.

**Length**  
Fullword.

The name of a fullword in which the service returns the previous interrupt state, or -1 if the service did not complete successfully.

**Return\_code**  
Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the pthread\_setintr service stores the return code. The pthread\_setintr service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_setintr service can return the following value in the Return\_code parameter:

Return code	Explanation
<b>EINVAL</b>	One of the parameters contains a value that is not valid.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the pthread\_setintr service stores the reason code. The pthread\_setintr service returns Reason\_code only if Return\_value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Setting the interruptability state allows you to control when cancellation requests sent via the pthread\_cancel (BPX1PTB, BPX4PTB) service are handled.
2. The pthread\_setintr (BPX1PSI, BPX4PSI) and pthread\_intrtype (BPX1PST, BPX4PST) services establish three interruptability states:

**Disabled**

Cancellation requests are left pending.

**Controlled**

Cancellation requests are left pending until the next cancellation point is reached. Cancellation points are defined as when the pthread\_testintr service is invoked (BPX1PTI, BPX4PTI) or a thread is placed in an unbounded wait during a call to a z/OS UNIX service. Some examples of these types of calls are

- [“close \(BPX1CLO, BPX4CLO\) — Close a file ” on page 121](#)
- [“cond\\_wait \(BPX1CWA, BPX4CWA\) — Suspend a thread for an event ” on page 136](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“pause \(BPX1PAS, BPX4PAS\) — Suspend a process pending a signal” on page 539](#)
- [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread” on page 592](#)
- [“pthread\\_testintr \(BPX1PTI, BPX4PTI\) — Cause a cancellation point to occur” on page 618](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered” on page 851](#)
- [“sigwait \(BPX1SWT, BPX4SWT\) — Wait for a signal ” on page 857](#)
- [“sleep \(BPX1SLP, BPX4SLP\) — Suspend execution of a process for an interval of time ” on page 859](#)
- [“tcdrain \(BPX1TDR, BPX4TDR\) — Wait until output has been transmitted ” on page 918](#)
- [“tcsetattr \(BPX1TSA, BPX4TSA\) — Set the attributes for a terminal ” on page 937](#)

- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

### Asynchronous

Cancellation request can be delivered at any time.

3. The default interrupt state for newly created threads and the initial thread is PTHREAD\_INTR\_ENABLE#.
4. The default interrupt type for newly created threads and the initial thread is PTHREAD\_INTR\_CONTROLLED#.
5. The interruption types of controlled and asynchronous are set with pthread\_intrtype (BPX1PST, BPX4PST). See [“pthread\\_setintrtype \(BPX1PST, BPX4PST\) — Examine and change the interrupt type” on page 613](#). These states are acted upon only if thread interruption is enabled. If a cancellation request is pending and the interrupt state or type is set to allow asynchronous cancellation requests, the thread is canceled before control is returned to the invoker.
6. See [Appendix I, “Optimizing performance using process- and thread-level information,” on page 1431](#).

### Related services

- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#)
- [“pthread\\_setintrtype \(BPX1PST, BPX4PST\) — Examine and change the interrupt type” on page 613](#)
- [“pthread\\_testintr \(BPX1PTI, BPX4PTI\) — Cause a cancellation point to occur” on page 618](#)

### Characteristics and restrictions

None.

### Examples

See [“BPX1PSI \(pthread\\_setintr\) example ” on page 1272](#) and [“BPX4PSI \(pthread\\_setintr\) example” on page 1364](#).

## pthread\_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type

---

### Function

The pthread\_setintrtype callable service sets the specified interruptability type of the calling thread and atomically returns the previous interruptability type.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PST):</b>	31-bit
<b>AMODE (BPX4PST):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

<b>Operation</b>	<b>Environment</b>
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1PST, (Interrupt_type,  
               Return_value,  
               Return_code,  
               Reason_code)
```

AMODE 64 callers use BPX4PST with the same parameters.

**Parameters**

**Interrupt\_type**  
Supplied parameter

**Type:**  
Structure

**Length:**  
Fullword

The name of a fullword containing a numeric value identifying the interrupt type to be set. The following constants, which are defined in BPXYCONS, define the valid states.

Constant	Description
<b>PTHREAD_INTR_ASYNCHRONOUS#</b>	When interruptability is enabled and the interruptability type is set to PTHREAD_INTR_ASYNCHRONOUS#, cancelation requests can be acted upon at any time.
<b>PTHREAD_INTR_CONTROLLED#</b>	When interruptability is enabled and the interruptability type is set to PTHREAD_INTR_CONTROLLED#, cancelation requests are held pending until a cancelation point is reached. See the usage notes for a definition of cancelation points.

**Return\_value**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword to which the service returns the previous interrupt type, or -1 if the service did not complete.

**Return\_code**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the pthread\_setintrtype service stores the return code. The pthread\_setintrtype service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The pthread\_setintrtype service can return the following value in the Return\_code parameter:

Return code	Explanation
EINVAL	One of the parameters contains a value that is not valid.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the pthread\_setintrtype service stores the reason code. The pthread\_setintrtype service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### Usage notes

1. The default interrupt type for newly created threads and the initial thread is PTHREAD\_INTR\_CONTROLLED#. If a cancelation request is pending and the interrupt state is set to PTHREAD\_INTR\_ASYNCHRONOUS#, the cancelation request is acted upon before control is returned to the invoker.
2. For more information about controlling cancelation requests, see the usage notes for [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state”](#) on page 610.
3. See [Appendix I, “Optimizing performance using process- and thread-level information,”](#) on page 1431.

### Related services

- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread”](#) on page 579
- [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state”](#) on page 610
- [“pthread\\_testintr \(BPX1PTI, BPX4PTI\) — Cause a cancellation point to occur”](#) on page 618

### Characteristics and restrictions

None.

### Examples

See [“BPX1PST \(pthread\\_setintrtype\) example”](#) on page 1272.

## pthread\_tag\_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data

---

### Function

The pthread\_tag\_np service sets, queries, or both sets and queries the 65 bytes of thread tag data that is associated with the caller's thread.

## Requirements

### Operation

#### Authorization:

Supervisor state or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1PTT):

31-bit

#### AMODE (BPX4PTT):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTT, (New_Tag_Length,
               New_Tag_Ptr,
               Old_Tag_Length,
               Old_Tag_Ptr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PTT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### New\_Tag\_Length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains either 0 or the length of the new tag data that is pointed to by New\_Tag\_Ptr. If New\_Tag\_Length contains 0 and New\_Tag\_Ptr contains a nonzero value, the caller's thread tag data is cleared. If New\_Tag\_Ptr contains a nonzero value, New\_Tag\_Length must be in the range of 0 to 65. See the usage notes for more details.

### New\_Tag\_Ptr

Supplied parameter

#### Type:

Pointer

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains either 0 or the address of a location that contains the new thread tag data. If New\_Tag\_Ptr contains 0, the caller's thread tag data is left unchanged. See the usage notes for more details.

### Old\_Tag\_Length

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_tag\_np service returns the length of the old (current) thread tag data that is returned to the caller.

**Old\_Tag\_Ptr**

Returned parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains either 0 or the address of a 65-byte area in which the pthread\_tag\_np service returns the old (current) thread tag data. If Old\_Tag\_Ptr contains 0, no thread tag data is returned to the caller and Old\_Tag\_Length remains unchanged. See the usage notes for more details.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_tag\_np service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_tag\_np service stores the return code. The pthread\_tag\_np service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The pthread\_tag\_np service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	One or more of the following conditions were detected: <ul style="list-style-type: none"><li>• All or part of the location that is specified by New_Tag_Ptr and New_Tag_Length was not addressable by the caller.</li><li>• All or part of the 66 bytes at the location that is specified by Old_Tag_Ptr was not addressable by the caller.</li></ul> The following reason codes can accompany the return code: JRNewLocationErr, or JROldLocationErr.
<b>EINVAL</b>	New_Tag_Ptr was nonzero, but New_Tag_Length was not in the range of 0 to 65. The following reason code can accompany the return code: JRNewLenBad.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the pthread\_tag\_np service stores the reason code. The pthread\_tag\_np service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If New\_Tag\_Ptr contains a nonzero value and New\_Tag\_Length contains 0, the caller's thread tag data is cleared.
2. If New\_Tag\_Ptr contains 0, the caller's thread tag data is left unchanged and the value specified by New\_Tag\_Length is not validity checked.
3. If the caller attempts to query the thread tag data and the tag data has never been set or was cleared, no data is stored at the location that is specified by Old\_Tag\_Ptr and Old\_Tag\_Length is set to 0.
4. If New\_Tag\_Ptr is nonzero, Tag\_Length must be in the range of 0 to 65. If it is not within range, the tag data is left unchanged and the pthread\_tag\_np service is unsuccessful.
5. Thread tag data is displayed with the DISPLAY OMVS command when 'PID=' option is specified. The thread tag data should be printable (EBCDIC) data.
6. When Old\_Tag\_Ptr is nonzero and the caller's thread has tag data associated with it (previously set and not cleared), the pthread\_tag\_np service stores the tag data (left justified) at the location that is specified by the caller, and Old\_Tag\_Length contains the length of the data that is stored.

**Related services**

None.

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1PTT \(pthread\\_tag\\_np\) example”](#) on page 1275.

## pthread\_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur

---

**Function**

The pthread\_testintr callable service causes a cancellation point to occur. If a cancellation request is pending, the cancellation request is acted upon before pthread\_testintr returns.

**Requirements**

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1PTI)</b>	31-bit



<b>Operation</b>	<b>Environment</b>
<b>AMODE (BPX4PTI)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTI, (Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PTI with the same parameters.

## Parameters

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which pthread\_testintr returns a 0 if the thread did not have any pending cancellation requests, or -1 if pthread\_testintr did not complete (the cancellation request was not tested).

### Return\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the pthread\_testintr service stores the return code. The pthread\_testintr service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

### Reason\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the pthread\_testintr service stores the reason code. The pthread\_testintr service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. If a cancellation request is pending when this service is requested, control is not returned.
2. Calling the pthread\_testintr service does not affect the interrupt state or type.
3. For more information about this service, see the usage notes for [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state”](#) on page 610.

## Related services

- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread”](#) on page 579
- [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state”](#) on page 610
- [“pthread\\_setintrtype \(BPX1PST, BPX4PST\) — Examine and change the interrupt type”](#) on page 613

## Characteristics and restrictions

None.

## Examples

See [“BPX1PTI \(pthread\\_testintr\) example”](#) on page 1273.

# ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging

---

## Function

The ptrace callable service provides information about another process and controls its running. Use this service in debugger programs to do breakpoint debugging.

## Requirements

Operation	Environment
<b>Authorization:</b>	Problem Program, PSW key 8
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1PTR):</b>	31-bit
<b>AMODE (BPX4PTR):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1PTR, (Request,
               Process,
               Address,
               Data,
               Buffer,
               Return_value,
```

```
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4PTR with the same parameters. The Address, Data, and Buffer parameters are doublewords.

## Parameters

### Request

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains one of the integer values that indicates the function requested. The request integer values are defined in the BPXYPTRC macro. See [“BPXYPTRC — Map parameters for ptrace” on page 1127](#).

### Process

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the process identifier of the process that is the target of the ptrace call, or 0 for the PT\_TRACE\_ME, PT\_EXTENDED\_EVENT, and PT\_RECOVER requests.

### Address

Supplied parameter

**Type:**

Address or Integer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains a value that is identified by the option selected for the Request parameter. For a mapping of this parameter to the Request parameter options, see [Table 24 on page 628](#).

### Data

Supplied parameter

**Type:**

Integer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains a value that is identified by the option selected for the Request parameter. For a mapping of this parameter to the Request parameter options, see [Table 24 on page 628](#).

### Buffer

Supplied parameter

**Type:**

Address or Integer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains a value that is identified by the option selected for the Request parameter. For a mapping of this parameter to the Request parameter options, see [Table 24 on page 628](#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the ptrace service returns 0; the requested value if the request is successful; or -1 if it is not successful. For more information about values that are returned for specific requests, see [Table 25 on page 631](#). A value of -1 is sometimes returned when the request is successful. For example, if a general-purpose register contains a value of -1, a PT\_READ\_GPR request returns this value in the Return\_value parameter.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the ptrace service stores the return code. The ptrace service always returns Return\_code, even if Return\_value is not -1. A Return\_code of 0 is returned for successful completion. For a list of return code values, see [Return codes \(errno\) in z/OS UNIX System Services Messages and Codes](#) For a mapping of these values to the various requests, see [Table 25 on page 631](#). The ptrace service can return one of the following values in the Return\_code parameter:

*Table 21. Return codes for ptrace*

Return_code	Explanation
EAGAIN	One or more resources are temporarily unavailable. Reissue the request at a later time.
ECHILD	The debugged process ended while a ptrace service request was running.
EFAULT	An address in the caller's process is incorrect. Possible reason codes are Subaddress and JRptInvDbrAddress.
EINTR	The ptrace service request was interrupted by a signal for the caller.

Table 21. Return codes for ptrace (continued)

Return_code	Explanation
<b>EINVAL</b>	<p>The request was not accepted, for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• The length is larger than the maximum defined length. The maximum defined length is defined in the BPXYPTRC macro.</li> <li>• The length of the area that is to contain the results of a PT_LDINFO, PT_LD64INFO, PT_EXPLAIN, or PT_THREAD_INFO request (the return information buffer) is too small to contain all the required information. For PT_LDINFO, PT_LD64INFO, and PT_EXPLAIN, increase the length up to the maximum defined length and reissue the request. For PT_THREAD_INFO, the required buffer length is returned. Reissue the request, using this returned buffer length. See Table 24 on page 628 for more information (the required length is returned to the Destination Address).</li> <li>• For the PT_CAPTURE request, the input address that is to be captured is not on a page boundary (4K).</li> <li>• For the PT_UNCAPTURE request, the input captured buffer address is not an address that was previously returned from a successful PT_CAPTURE request.</li> <li>• For the PT_BLOCKEDREQ request, some of the requests might not have completed successfully. The Reason_code is set to JRPtSomeBlkedFailed. Check the PtBRStatus field of the PtBRInfo block for each blocked request to determine which have failed.</li> <li>• For the PT_LDINFO request, some RMODE64 modules were found. The Reason_code is set to JRPtLDRMODE64Loadmod. Information on all other loaded modules will still be returned, but PT_LD64INFO must be run to obtain complete information on all loaded modules.</li> </ul> <p>Possible reason codes are: JRPtLDBufferTooSmall, JRBuffTooSmall, JRNotPage, JRPtBufNotFound, JRPtLDRMODE64Loadmod, JRPtInvLength.</p>

Table 21. Return codes for ptrace (continued)

Return_code	Explanation
<b>EIO</b>	<p>The request was not accepted for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• The caller is not running with PSW key 8.</li> <li>• An incorrect Request was specified.</li> <li>• For a PT_TRACE_ME or PT_ATTACH request, the target process is already being debugged. For a PT_REATTACH or PT_REATTACH2 request, the target process is not already being debugged.</li> <li>• For a PT_DETACH, PT_CONTINUE or PT_THREAD_SIGNAL request, the signal number that was supplied in the Data parameter is not a valid signal number.</li> <li>• An address in the target process is not valid.</li> <li>• A register number for a PT_READ_GPR, PT_WRITE_GPR, PT_READ_FPR, PT_WRITE_FPR, PT_READ_VR, or PT_WRITE_VR request is not defined. The register numbers are defined in the BPXYPTRC macro.</li> <li>• An attempt was made to store into a control register using the PT_WRITE_GPR request.</li> <li>• An attempt was made to store into the left half of the PSW using the PT_WRITE_GPR request.</li> <li>• The user area offset that was supplied with the PT_READ_U request is incorrect.</li> <li>• For the PT_TRACE_ME request, the parent of the debugged process (that is, the debugger) has ended.</li> <li>• For the PT_REATTACH or PT_REATTACH2 request, the original debugger has ended.</li> <li>• For the PT_THREAD_WRITE_FOCUS, PT_THREAD_HOLD, PT_THREAD_MODIFY and PT_THREAD_SIGNAL requests, the thread ID that was supplied is not valid.</li> <li>• For the PT_EXPLAIN request, an extended ptrace event is not in progress.</li> <li>• For the PT_EVENTS request, an attempt was made to add more extended events than the maximum number of events that was specified on the PT_EVENTS request.</li> <li>• The request is not supported while it is stopped for a local fork child, or for an extended event.</li> <li>• For the PT_CAPTURE request, the target process is running in a TSO address space.</li> </ul> <p>Possible reason codes are: JRPtAttemptedCRStore, JRPtAttemptedPSW0Store, JRPtDbdParentTerm, JRPtDbrPidNotFound, JRPtDbrZombie, JRPtInvCallingMode, JRPtInvDbdAddress, JRPtInvFPRNumber, JRPtInvGPRNumber, JRPtInvNumberThreads, JRPtInvPtraceState, JRPtInvRequest, JRPtInvSignalNumber, JRPtInvUAreaOffset, JRPtOldDbrPidNotFound, JRPtThreadTerm, JRPtLightWeightTHID, JRPtThreadNotFound, JRPtTSO, JRPtRequestDenied, JRPtAsyncThread, JRPtNotXtdEvent, JRPtTooManyEvents, JRPTInvVRNumber.</p>

Table 21. Return codes for ptrace (continued)

Return_code	Explanation
<b>EMVSSAF2ERR</b>	For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the caller does not have the appropriate privileges to debug the target process. For information about appropriate privileges, see <a href="#">“Authorization”</a> on page 8.
<b>ENOMEM</b>	There is not enough storage available to satisfy a PT_CAPTURE request.
<b>EPERM</b>	<p>Permission to issue the request is denied for one of the following reasons:</p> <ul style="list-style-type: none"> <li>For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the target process is restricted from being debugged.</li> </ul> <p>For PT_REATTACH or PT_REATTACH2, it is more likely that EIO will be returned, because the target process is not already being debugged. However, in the unlikely event that a restricted process successfully issues a PT_TRACE_ME request, a PT_REATTACH or PT_REATTACH2 could return EPERM.</p> <p>If either of the following is true, the target process is restricted:</p> <ul style="list-style-type: none"> <li>The target process is a system address space. For more information about system address spaces, see <a href="#">“MVS-related information”</a> on page 644.</li> <li>The target process is the INIT process, indicated by a process ID (PID) value of 1.</li> </ul> <ul style="list-style-type: none"> <li>For the PT_READ_xxx, PT_WRITE_xxx, PT_CONTINUE (to continue at another address), PT_REGSET and PT_LDINFO requests, the target process is currently running in supervisor state.</li> </ul> <p>The following reason codes can accompany the return code: JRPtRestrictedProcess, JRPtEdIsAuthorized.</p>
<b>ESRCH</b>	<p>The request was not accepted, for one of the following reasons:</p> <ul style="list-style-type: none"> <li>For all requests other than PT_TRACE_ME, PT_ATTACH, PT_REATTACH, PT_REATTACH2, PT_EXTENDED_EVENT, and PT_RECOVER, the target process is not being debugged.</li> <li>For all requests other than PT_TRACE_ME, PT_ATTACH, PT_REATTACH, PT_REATTACH2, PT_EXTENDED_EVENT, and PT_RECOVER, the target process is not stopped for a ptrace service event.</li> <li>For all requests other than PT_TRACE_ME, PT_EXTENDED_EVENT, and PT_RECOVER, the target process ID is incorrect.</li> <li>For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the target debugged process is the same as the debugger process.</li> <li>For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the target debugged process is the parent of the debugger process.</li> </ul> <p>The following reason codes can accompany the return code: JRPtDbdEqualsDbr, JRPtDbdPidNotFound, JRPtProcessNotPtraced, JRPtProcessNotStopped, JRPtDbrParentEqualsDbd.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the ptrace service stores the reason code. The ptrace service always returns Reason\_code, even if Return\_value is not -1. A Reason\_code of 0 is returned for successful completion. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF ptrace Authority Check service return and reason code values, see [Table 22 on page 626](#):

*Table 22. RACF return and reason codes for the ptrace authority check service*

Return code	Reason code	Explanation
8	4	The caller is not authorized to attach to the target process
8	12	Internal error during RACF processing

### Constant options for the ptrace request parameter

Table 23 on page 626 shows the constant options that you can select for the Request parameter. See “BPXYPTRC — Map parameters for ptrace” on page 1127 for the constant definitions.

<i>Table 23. Constant options for the ptrace request parameter</i>	
Constant	Explanation
<b>PT_ATTACH</b>	Enable a target process to be debugged with the ptrace service.
<b>PT_CAPTURE</b>	Capture one or more pages of storage in the target debugged process into a buffer in the caller's address space.
<b>PT_CONTINUE</b>	Continue running the debugged process.
<b>PT_DETACH</b>	Disable debugging for the target process.
<b>PT_EVENTS</b>	Enable or disable reporting for an extended event.
<b>PT_EXPLAIN</b>	Return additional information about an extended event.
<b>PT_EXTENDED_EVENT</b>	Notify the debugger of an extended event.
<b>PT_KILL</b>	End the debugged process.
<b>PT_LDINFO</b>	Return information about modules that were loaded by the debugged process.
<b>PT_LD64INFO</b>	Return information about the RMODE64 modules that were loaded by the debugged process.
<b>PT_MULTI</b>	Turn multiprocess debugging mode on or off. For information about multiprocess debugging, see “Multiprocess debugging mode” on page 641.
<b>PT_BLOCKREQ</b>	Several Ptrace request types are blocked together into a single Ptrace call.
<b>PT_READ_BLOCK</b>	Read a block of storage.
<b>PT_READ_D</b>	Return a fullword of data from a specified address in the debugged process. This request reads program data.
<b>PT_READ_FPR</b>	Return the value of a floating-point register.



Table 23. Constant options for the ptrace request parameter (continued)

Constant	Explanation
<b>PT_READ_GPR</b>	Return the value of a general-purpose or machine-control register. The value includes the PSW and control registers, as well as general-purpose registers.
<b>PT_READ_GPRH</b>	Read a specific general-purpose high register.
<b>PT_READ_I</b>	Return a fullword of data from a specified address in the debugged process. This request reads program instructions.
<b>PT_READU</b>	Return the value of a fullword of control information from the user area in the debugged process. For more information, see <a href="#">“User area description”</a> on page 642.
<b>PT_READ_VR</b>	Return the 16-byte value of a vector register.
<b>PT_WRITE_VR</b>	Change the 16-byte value of a vector register.
<b>PT_REATTACH</b>	Enable a target process to be debugged with the ptrace service by a new debugger. The relationship between the target process and its original debugger is removed.
<b>PT_REATTACH2</b>	Enable a target process to be debugged with the ptrace service by a new debugger. The relationship between the target process and its original debugger is removed. This request is an extension of the PT_REATTACH request, and must be used by a debugger to deal with the local fork child environment. For more information, see <a href="#">“Attaching to a process for debugging”</a> on page 633.
<b>PT_RECOVER</b>	Notify the debugger of a program check interrupt or abnormal end. For more information, see <a href="#">“Handling a program check or abend in a debugged process”</a> on page 636.
<b>PT_REGHSET</b>	Read all of the general-purpose high registers.
<b>PT_REGSET</b>	Return the values of all general-purpose registers.
<b>PT_THREAD_HOLD</b>	Hold or unhold a thread in the debugged process.
<b>PT_THREAD_INFO</b>	Return kernel information on all threads in the debugged process.
<b>PT_THREAD_MODIFY</b>	Modify a thread's kernel information.
<b>PT_THREAD_READ_FOCUS</b>	Return the current focus thread ID.
<b>PT_THREAD_SIGNAL</b>	Queue a signal to a thread in the debugged process.
<b>PT_THREAD_WRITE_FOCUS</b>	Change the current focus thread ID.
<b>PT_TRACE_ME</b>	Enable the calling process to be debugged with the ptrace service.

<i>Table 23. Constant options for the ptrace request parameter (continued)</i>	
Constant	Explanation
<b>PT_UNCAPTURE</b>	Free one or all buffers that contain captured storage from previous PT_CAPTURE requests.
<b>PT_WRITE_BLOCK</b>	Change the contents of a block of storage.
<b>PT_WRITE_D</b>	Change a fullword of data at a specified address in the debugged process. This request changes program data.
<b>PT_WRITE_FPR</b>	Change the value of a floating-point register.
<b>PT_WRITE_GPR</b>	Change the value of a general-purpose or machine-control register. The value includes the PSW and control registers, as well as general-purpose registers.
<b>PT_WRITE_GPRH</b>	Write to a specific general-purpose high register.
<b>PT_WRITE_I</b>	Change a fullword of data at a specified address in the debugged process. This request changes program instructions.

## Parameter attributes for request options

Table 24 on page 628 shows the ptrace service options for the Request parameter. For each option, the meanings of the Address, Data, and Buffer parameters are shown. Explanations of the terms in the table follow the table:

<i>Table 24. Parameter attributes for request options</i>			
Request options	Address	Data	Buffer
<b>PT_ATTACH</b>	0	0	0
<b>PT_CAPTURE</b>	Capture Address	Capture Length	0
<b>PT_CONTINUE</b>	1 = Continue from where process stopped  Not 1 = Continue Address	0 = No signal  Not 0 = Signal Number	0
<b>PT_DETACH</b>	0	0 = No signal  Not 0 = Signal Number	0
<b>PT_EVENTS</b>	Extended Event Id	0 = Disable re- porting this event  Not 0 = Enable reporting this event	Maximum Events
<b>PT_EXPLAIN</b>	Buffer Address (destination)	Length	0
<b>PT_EXTENDED_EVENT</b>	GIParm Address	Extended Event ID	Destination Address (4 bytes)
<b>PT_KILL</b>	0	0	0
<b>PT_LDINFO</b>	Buffer Address (Destination)	Length	0

Table 24. Parameter attributes for request options (continued)

Request options	Address	Data	Buffer
<b>PT_LD64INFO</b>	Buffer Address (Destination)	Length	0
<b>PT_MULTI</b>	0	0 = Reset multi- process mode  Not 0 = Set multiprocess mode	0
<b>PT_BLOCKREQ</b>	Buffer Address (source / destination)	Length	Buffer Address (destination)
<b>PT_READ_BLOCK</b>	Debugged Address	Length	Buffer Address
<b>PT_READ_D</b>	Debugged Address	0	0
<b>PT_READ_FPR</b>	Destination Address	Register Number	0
<b>PT_READ_GPR</b>	Register Number	0	0
<b>PT_READ_GPRH</b>	Register Number	0	0
<b>PT_READ_I</b>	Debugged Address	0	0
<b>PT_READ_U</b>	Target Offset	0	0
<b>PT_READ_VR</b>	Destination address	Register number	0
<b>PT_REATTACH</b>	0	0	0
<b>PT_REATTACH2</b>	0	0	Destination Address
<b>PT_RECOVER</b>	PCParm Address	0	0
<b>PT_REGHSET</b>	Destination Address	0	0
<b>PT_REGSET</b>	Destination Address	0	0
<b>PT_THREAD_HOLD</b>	Thread ID Address	0 = Unhold thread  Not 0 = Hold thread	0
<b>PT_THREAD_INFO</b>	Buffer Address	Length	Destination Address
<b>PT_THREAD_MODIFY</b>	Thread ID Address	0	Source Address
<b>PT_THREAD_READ_FOCUS</b>	Thread ID Address	0	0
<b>PT_THREAD_SIGNAL</b>	Thread ID Address	Signal Number	0
<b>PT_THREAD_WRITE_FOCUS</b>	Thread ID Address	0	0
<b>PT_TRACE_ME</b>	0	0	0
<b>PT_UNCAPTURE</b>	0 = Free all buffers  Not 0 = Capture Buffer	0	0
<b>PT_WRITE_BLOCK</b>	Debugged Address	Length	Buffer Address
<b>PT_WRITE_D</b>	Debugged Address	Integer Value	0
<b>PT_WRITE_FPR</b>	Source Address	Register Number	0
<b>PT_WRITE_GPR</b>	Register Number	Register Value	0
<b>PT_WRITE_GPRH</b>	Register Number	Register Value	0
<b>PT_WRITE_I</b>	Debugged Address	Integer Value	0
<b>PT_WRITE_VR</b>	Source address	Register number	0

**Buffer Address**

The name of a fullword that contains an address in the caller's process where either:

- The results of the request are to be placed
- The source information for the request is to be obtained

The size of the buffer is specified with the Length parameter.

**Capture Address**

The name of a fullword that contains an address in the target process that is to be captured into a buffer in the caller's address space. This address must be on a page boundary (4K).

**Capture Buffer**

The name of a fullword that contains an address in the caller's process that represents a captured storage buffer. This address must have been previously returned to the caller on a PT\_CAPTURE request.

**Capture Length**

The name of a fullword that contains the length of the storage that is to be captured. There is no need to round this length up to the size of a page.

**Continue Address**

The name of a fullword that contains an address in the target process from which the debugged program is to continue running. The address must include the addressing mode (AMODE) as the high-order bit. A high-order bit of 0 indicates a 24-bit AMODE; a high-order bit of 1 indicates a 31-bit AMODE.

The PT\_CONTINUE request can indicate a value of 1 instead of an address that indicates where continuation should begin. This value, which is defined in the BPXYPTRC macro, indicates that the program should continue from where it stopped.

**Debugged Address**

The name of a fullword that contains an address in the target process.

**Destination Address**

The name of a fullword that contains an address in the caller's process at which the results of the request are to be placed. The size of the destination area is defined by the request type.

**Extended Event ID**

The name of a fullword that contains an extended event ID.

**Integer Value**

The name of a fullword that contains the value that is to be placed at the Debugged Address location.

**GIParm Address**

The name of a fullword that contains the address of the generic interface parameters. For more information, see [“Handling extended events in a debugged process”](#) on page 638.

**Length**

The name of a fullword that contains the length that is associated with the Buffer Address. The maximum length value is defined in the BPXYPTRC macro, except for the PT\_THREAD\_INFO request.

**Maximum Events**

The name of a fullword that contains the maximum number of extended events that will be added using the PT\_EVENTS request. This is required only for the first issuance of PT\_EVENTS, but it can be specified on all issuances.

**PCParm Address**

The name of a fullword that contains the address of the program check parameters. For more information, see [“Handling a program check or abend in a debugged process”](#) on page 636.

**Register Number**

The name of a fullword that contains a defined register number. The register numbers are defined in the BPXYPTRC macro.

**Register Value**

The name of a fullword that contains the register value that is to be placed in the Register Number in the target process.

**Signal Number**

The name of a fullword that contains the signal number that is to be sent to the target debugged process or thread. The signal numbers are defined in the BPXYSIGH macro.

**Source Address**

The name of a fullword that contains an address in the caller's process where the source information for the request is to be obtained. The size of the source area is defined by the request type.

**Target Offset**

The name of a fullword that contains an offset into the user area in the target process. The user area contains control information. For a description of the user area, see [“User area description” on page 642](#).

**Thread ID Address**

The name of a fullword that contains an address in the caller's process where either:

- The target thread ID is to be placed
- The target thread ID is to be obtained

The length of the thread ID is 8 bytes.

**Return values and return codes for request options**

Table 25 on page 631 shows the ptrace service requests. For each request, the value that is returned in the Return\_value parameter is shown. Possible values returned in the Return\_code parameter are also shown.

<i>Table 25. Return values and return codes for request options</i>		
<b>Request</b>	<b>Return_value</b>	<b>Return_code</b>
<b>(General)</b>	0	EFAULT, EIO, EMVSERR
<b>PT_ATTACH</b>	0	EAGAIN, ECHILD, EIO, EMVSSAF2ERR, EPERM, ESRCH
<b>PT_BLOCKREQ</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EINVAL, EPERM, ESRCH
<b>PT_CAPTURE</b>	Capture buffer address	EINVAL, EIO, ENOMEM
<b>PT_CONTINUE</b>	Value of Data parameter	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_DETACH</b>	0	EAGAIN, ECHILD, EINTR, EIO, ESRCH
<b>PT_EVENTS</b>	0	ESRCH
<b>PT_EXPLAIN</b>	0	EFAULT, EIO, ESRCH
<b>PT_EXTENDED_EVENT</b>	0	EFAULT
<b>PT_KILL</b>	0	EAGAIN, ECHILD, EINTR, ESRCH
<b>PT_LDINFO</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EINVAL, EPERM, ESRCH
<b>PT_LDINFO64</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EINVAL, EPERM, ESRCH
<b>PT_MULTI</b>	0	ESRCH
<b>PT_READ_BLOCK</b>	Value of Data parameter	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EINVAL, EPERM, ESRCH

<i>Table 25. Return values and return codes for request options (continued)</i>		
<b>Request</b>	<b>Return_value</b>	<b>Return_code</b>
<b>PT_READ_D</b>	Fullword value	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_READ_FPR</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH
<b>PT_READ_GPR</b>	Register contents	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_READ_GPRH</b>	Register contents	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_READ_I</b>	Fullword value	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_READ_VR</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH
<b>PT_READ_U</b>	Fullword value	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_REATTACH</b>	0	EAGAIN, ECHILD, EIO, EMVSSAF2ERR, EPERM, ESRCH
<b>PT_REATTACH2</b>	0	EAGAIN, ECHILD, EIO, EMVSSAF2ERR, EPERM, ESRCH
<b>PT_RECOVER</b>	0	EFAULT
<b>PT_REGHSET</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EPERM, ESRCH
<b>PT_REGSET</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EPERM, ESRCH
<b>PT_THREAD_HOLD</b>	0	EFAULT, EIO, ESRCH
<b>PT_THREAD_INFO</b>	0	EFAULT, EINVAL, ESRCH
<b>PT_THREAD_MODIFY</b>	0	EFAULT, EINVAL, EIO, ESRCH
<b>PT_THREAD_READ_FOCUS</b>	0	EFAULT, ESRCH
<b>PT_THREAD_SIGNAL</b>	0	EFAULT, EIO, ESRCH
<b>PT_THREAD_WRITE_FOCUS</b>	0	EFAULT, EIO, ESRCH
<b>PT_TRACE_ME</b>	0	EAGAIN, EIO
<b>PT_UNCAPTURE</b>	0	EINVAL
<b>PT_WRITE_BLOCK</b>	Value of Data parameter	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EINVAL, EPERM, ESRCH
<b>PT_WRITE_D</b>	0	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_WRITE_FPR</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH
<b>PT_WRITE_GPR</b>	Value of Data parameter	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH

Table 25. Return values and return codes for request options (continued)

Request	Return_value	Return_code
<b>PT_WRITE_GPRH</b>	Value of Data parameter	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_WRITE_I</b>	0	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
<b>PT_WRITE_VR</b>	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH

## Starting a process in debugging mode

Typically, a debugger program starts a process to be debugged by calling the fork service to create a child copy of the debugger program. The child then calls the ptrace service with a PT\_TRACE\_ME request. This puts the child process into debugging mode. Next, the child calls the exec service to run the program to be debugged.

The PT\_TRACE\_ME request, along with PT\_EXTENDED\_EVENT and PT\_RECOVER, is issued from the process to be debugged. All other ptrace service requests are issued from the debugger. It is also assumed that the parent of the process issuing a PT\_TRACE\_ME request is the debugger.

## Attaching to a process for debugging

The ptrace service also provides a means for a debugger program to debug an already running, possibly unrelated, process. To do this, the debugger calls ptrace with a PT\_ATTACH service request. There are certain restrictions on which processes can be attached (see [“Characteristics and restrictions” on page 643](#)). The caller must have the appropriate privileges (see [“Authorization” on page 8](#)) to attach to a running process.

The PT\_REATTACH request performs a similar function, but is intended to be used in a situation where another debugger process is already attached to the target process. The PT\_REATTACH request causes the relationship between the other debugger and the target process to be severed. The caller of PT\_REATTACH becomes the new debugger associated with the target process. The PT\_REATTACH2 request is identical to PT\_REATTACH except in one respect. PT\_REATTACH2 provides the address of an area in which return information concerning the reattach can be placed. If the PT\_REATTACH2 request is issued against the child process that was created with an attach\_exec or attach\_execmvs service (a local fork child), the return information is nonzero, indicating to the debugger that alternate reattach processing is required. Otherwise, the return information is zero, telling the debugger that reattach processing should be the same as for PT\_REATTACH.

This is required because of the restricted nature of the local fork child environment. Normally, after a reattach to a fork child, the debugger can continue issuing ptrace requests, because the child is a copy of the parent and is already stopped for a ptrace event. However, a local fork child is not a copy of the parent, and in fact most ptrace requests fail if issued to a local fork child, because the target program that is specified on the attach\_exec or attach\_execmvs service has not yet been loaded for execution. The alternate processing required by the debugger for this environment is to issue a PT\_CONTINUE request, followed by a wait(). This causes the local fork child to continue until the next ptrace event, which will be an exec event for the attach\_exec service. At this point, the target program is loaded, and the debugger can continue issuing ptrace requests in this valid environment.

**Note:** ptrace has no way to know whether the target of the PT\_REATTACH2 is the local fork parent or the local fork child. It is the debugger's responsibility to issue the request for the local fork child only. The result of issuing this request for the local fork parent is that control of the parent is lost; it will continue running until the next ptrace event.

Here is an example of using the PT\_REATTACH or PT\_REATTACH2 request: Debugger 1 is currently debugging program A in multiprocess mode. For information about multiprocess debugging, see [“Multiprocess debugging mode” on page 641](#). Program A uses the fork service to create a child process,

which becomes program B. Debugger 1 is informed of the fork from both the parent (program A) and child (program B) processes. Debugger 1 uses the fork service to create a new debugger, which becomes debugger 2. Debugger 2 then uses the PT\_REATTACH request to associate itself with program B. At this point, debugger 1 is debugging program A, and debugger 2 is debugging program B.

The PT\_ATTACH, PT\_REATTACH, and PT\_REATTACH2 requests cause a SIGTRAP signal to be sent to the target process. This causes a ptrace service signal event to occur if no other event occurs naturally.

## Receiving notification of events in a debugged process

When a process has been placed into debugging mode by the PT\_TRACE\_ME, PT\_ATTACH, PT\_REATTACH, or PT\_REATTACH2 request, certain events in the debugged process cause the process to be placed into a stopped state and the debugger to be notified. The debugger must wait for these events by using the wait service. The following are the events of interest:

- A signal is received. The Status\_field parameter on the wait service issued by the debugger contains the signal number.
- An exec service is issued. The Status\_field parameter on the wait service issued by the debugger either contains the SIGTRAP signal number, if multiprocess debugging is not in effect; or indicates that the process stopped for an exec (WastStopFlagExec), if multiprocess debugging is in effect. For information about multiprocess debugging, see [“Multiprocess debugging mode”](#) on page 641. Also see [“BPXYWAST – Map the wait status word”](#) on page 1172 for a description of the Wast values.
- A fork service is issued and multiprocess debugging mode is in effect. The Status\_field parameter on the wait service issued by the debugger indicates that the process stopped for a fork (WastStopFlagFork).
- An attach\_exec or attach\_execmvs service call is issued and multiprocess debugging mode is in effect. The Status\_field parameter on the wait service issued by the debugger indicates that the process stopped for a local fork (WastStopFlagLocalFork).
- A spawn service call is issued and multiprocess debugging mode is in effect. The Status\_field parameter on the wait service issued by the debugger indicates that the process stopped for a fork (WastStopFlagFork) for the spawn parent, and for a local fork (WastStopFlagLocalFork) for the spawn child.
- An SVC 144 instruction is run. SVC 144 is used as a breakpoint by debugger programs. The debugger uses the ptrace service to store the SVC 144 instructions into the program at the appropriate breakpoints. The Status\_field parameter on the wait service that was issued by the debugger contains the SIGTRAP signal number.
- A program check or abnormal end is encountered. The debugger is notified only if the program check or abnormal end causes the ptrace service to be called with a PT\_RECOVER request. This is normally true for programs that detect the error, and that can provide the proper interface to the PT\_RECOVER request. An ESPIE routine is an example of this. The Status\_field parameter on the wait service that was issued by the debugger contains the appropriate signal number. For more information, see [“Handling a program check or abend in a debugged process”](#) on page 636.
- An extended event occurs and a generic debugger interface module issues the ptrace PT\_EXTENDED\_EVENT request. Extended events are enabled by using the PT\_EVENTS service request. Only those events thus enabled cause a ptrace extended event to occur. [“Handling extended events in a debugged process”](#) on page 638 provides more information about the generic debugger interface. The debugger must use the PT\_EXPLAIN request to obtain additional information about the extended event. The Status\_field parameter on the wait service that was issued by the debugger indicates that the process stopped for an extended event (WastStopFlagExtended).
- A loadhfs service is issued. The Status\_field parameter on the wait service issued by the debugger indicates that the process stopped because of a file system module load (WastStopFlagLoad).
- A deletehfs service is issued. The Status\_field parameter on the wait service that was issued by the debugger indicates that the process stopped because of a file system module delete (WastStopFlagDelete).



Table 26 on page 635 summarizes the events and the corresponding status reported to the debugger from the wait call:

<i>Table 26. Corresponding ptrace event and status that is reported to the debugger</i>	
<b>ptrace event</b>	<b>Debugger wait service Status_field parameter</b>
<b>Signal received</b>	Signal number
<b>exec service issued</b>	SIGTRAP signal number or WastStopFlagExec
<b>fork service issued</b>	WastStopFlagFork
<b>attach_exec or attach_execmvs service issued</b>	WastStopFlagLocalFork
<b>spawn service issued</b>	WastStopFlagFork (parent), WastStopFlagLocalFork (child)
<b>SVC 144 instruction performed</b>	SIGTRAP signal number
<b>Program check or abend encountered</b>	SIGILL, SIGSEGV, SIGFPE, or SIGABND signal number
<b>Extended event encountered</b>	WastStopFlagExtended

When a process has multiple threads, any thread that encounters one of the ptrace service events causes the process to enter a stopped state. This is accomplished by synchronously suspending all other threads in the process. The thread on which the event occurred is known as the *focus thread*. Because delays could occur between the time the focus thread encounters the ptrace event and the time all the nonfocus threads are suspended, one or more of these other threads could encounter the same or other ptrace events. For instance, several threads could reach a breakpoint in a routine that is common to them all. This creates a situation in which the focus thread is "in control" of ptrace processing, but other ptrace service events are pending. See “Working with threads in a debugged process” on page 635 for more information about handling threads in a debugged process.

While the debugged process is stopped for one of the foregoing events, the debugger can issue ptrace service requests to examine or modify registers, storage, and so on. Most ptrace service requests are issued while the debugged process is stopped for the ptrace service event. Examples are: PT\_LDINFO and PT\_READ\_U. An event ends when a PT\_CONTINUE, PT\_DETACH, or PT\_KILL request is issued. One exception to this is a PT\_CONTINUE request with a signal that stops the debugged process (for instance, SIGSTOP). In this case, the original event does not end until a PT\_CONTINUE is issued with either no signal or a SIGCONT. This is because of the ambiguous nature of a "continue and stop" request. The debugged process does not actually continue running until it is taken out of the stopped state, either explicitly by a PT\_CONTINUE with SIGCONT request, or implicitly by a PT\_CONTINUE with no signal.

## Working with threads in a debugged process

Several ptrace service requests can assist debuggers in handling multiple threads. The PT\_THREAD\_INFO service request returns a list of threads and kernel information about each thread, such as its state (active, dead, and so on) and kernel attributes. The PT\_THREAD\_READ\_FOCUS service request returns the current focus thread ID. These ptrace service requests allow the debugger to gather various thread-related information whenever the debugger is awoken for a ptrace service event.

Certain debugger objectives require exact control over which thread or threads are running at any given time. For example, if the debugger wants to single-step the focus thread, the focus thread must be the only thread in the target process that is running. If this is not so, unpredictable results could occur. The PT\_THREAD\_HOLD service request allows the debugger to selectively place any threads that are not in a dead state into a held state. When a thread is held, it does not run until it is released. The debugger could therefore use this request to hold all but the focus thread, and then single-step the focus thread by

inserting breakpoints after each program statement. The PT\_THREAD\_HOLD service request can also be used to release threads.

The debugger might also want to work with threads other than the current focus thread. An example might be if the current focus thread manipulates data in shared storage that is then acted upon by a different thread. In order to work with this other thread, the debugger must release the thread, and then shift focus to it by using the PT\_THREAD\_WRITE\_FOCUS service request. This request causes the specified thread to become the new focus thread if it is in an active, non-asynchronous state. Other ptrace service requests that read or write storage, registers, and other data always act against the current focus thread, so this is the means by which the debugger specifies which thread is the target of these other requests.

Signals can be sent to the debugged process using the PT\_CONTINUE and PT\_DETACH requests, as explained [“Resuming or detaching from a debugged process.”](#) on page 640. These signals are always directed at the current focus thread. The PT\_THREAD\_SIGNAL service request provides a means to send signals to individual threads that are not in a dead state. The signal that is specified on this request is generated for the target thread, but it is not delivered until the process is continued. A debugger can use this request to requeue signals that were pending on a thread when a ptrace event occurred. Normally, all signals (except those specified on PT\_CONTINUE or PT\_DETACH) that are pending on the focus thread are discarded when the process is continued. Using the signal information that is returned on the PT\_THREAD\_INFO request, the debugger could regenerate all pending signals by using the PT\_THREAD\_SIGNAL request. This is useful when changing thread focus; otherwise, any signals that were pending on nonfocus threads when the original ptrace event occurred might be lost.

**Note:** To ensure that pending signal information for the focus thread is not lost (because the signals were discarded), the PT\_THREAD\_INFO request should be the first ptrace request that is issued when the debugger gets notified of a ptrace service event.

Finally, the PT\_THREAD\_MODIFY service request allows the debugger to modify individual thread kernel information for a thread that is in a dead state. The input to this request is a single thread info array entry, as returned by the PT\_THREAD\_INFO request, modified as necessary with the required changes. The following information is allowed to be changed:

- Thread exit status for threads in a dead state (PtPtExitStatus)

## Determining modules loaded in a debugged process

If the debugger needs to determine the names and entry points of modules that are loaded into the debugged process, it uses the ptrace PT\_LDINFO or PT\_LD64INFO service request. A structure is returned to the debugger that contains information about loaded modules, including the name of the directory that contains the load module for each module loaded from the file system. (The directory name is not returned for modules loaded from MVS data sets.) One use for this information is to read the load module library file to obtain symbolic debugging information. The returned structure is defined in the BPXYPTRC macro.

If any above the bar modules are detected with PT\_LDINFO, they are excluded from the resulting data structure and an error code will be returned. Use PT\_LD64INFO instead of PT\_LDINFO because it will correctly return information about both above and below the bar modules that are currently loaded.

For more information about PT\_LDINFO or PT\_LD64INFO, see [“MVS-related information”](#) on page 644.

## Handling a program check or abend in a debugged process

When program checks or abnormal ends occur in a debugged process and are captured by the program's recovery routine (such as an ESPIE or ESTAE exit), the PT\_RECOVER request can be issued. This request allows the ptrace service to stop the process and notify the debugger that a program check or abnormal end has just occurred. The caller does not need to determine if the process is being debugged; it can issue the PT\_RECOVER request unconditionally. If the process is not being debugged, the returned information indicates to the caller that it can continue as it normally would. The returned information contains PtPICFlags, which are all zeros on return if the process is not being debugged.

However, if the process is being debugged, the ptrace service stops the process by sending an appropriate signal to the focus thread. The debugger (not the application) sets this signal with the `Status_field` parameter on the wait call. These are the possible signals that are used for program checks:

### **SIGILL**

Unpermitted operation, defined as one of the following:

- Operation exception
- Privileged operation exception
- Execute exception
- Specification exception

### **SIGSEGV**

Addressing error, defined as one of the following:

- Protection exception
- Addressing exception

### **SIGFPE**

Arithmetic error, defined as one of the following:

- Data exception
- Fixed-point overflow exception
- Fixed-point divide exception
- Decimal overflow exception
- Decimal divide exception
- Exponent overflow exception
- Exponent underflow exception
- Significance exception
- Floating-point divide exception

The interface to the `PT_RECOVER` request is the program check parameters structure, which is pointed to from the `Address` parameter. This structure contains pointers to environment information from the time of the program check or abnormal end. It also contains flags and information that is returned from the ptrace service. The program check parameters structure is defined in the `BPXYPTRC` macro. The ptrace service distinguishes between program checks and other abnormal ends. The presence of an abend code causes the interrupt to be interpreted as a `SIGABND` signal. The presence of an interrupt code without an abend code causes it to be interpreted as `SIGILL`, `SIGSEGV`, or `SIGFPE`, as appropriate. The following shows the information that must be present in the program check parameters structure on input to `PT_RECOVER`:

1. In all cases the following must be set:
  - `PtPICRegisters` = address of registers at time of error (0 - 15)
  - `PtPICPSW` = address of PSW at time of error
  - `PtPICFlags` = 0 (except `PtPICILCExists` if `PtPICILC` is set)
2. For program checks, the following must be set:
  - `PtPICIntCode` = program interrupt code
  - `PtPICAbendCode` = 0
3. For program checks, the following are optional:
  - `PtPICILC` = instruction length code
  - `PtPICILCExists` = flag set to 1 to indicate `PtPICILC` is valid
4. For non-program check abnormal ends, the following must be set:
  - `PtPICAbendCode` = abend code

- PtPICAbendReason = reason code
  - PtPICILCExists = flag set to 0 to indicate PtPICILC is not used
5. For any program check that is not specified in the foregoing list, SIGFPE is used. For abnormal ends (abends) SIGABND is used.

The environment information (registers and PSW) can be modified by the debugger by using appropriate ptrace service requests while the debugged process is stopped for the ptrace service event. When the program is continued with the PT\_CONTINUE or PT\_DETACH request, you need to ensure that any modifications are reflected in the operating environment when the program resumes control.

When the registers or the PSW are changed by the debugger, the appropriate flag in the program check parameters, as defined in the BPXYPTRC macro, is set to indicate this. Conversely, if the registers or PSW are not changed, the flag is not set. Thus the caller can test these flags to determine if changes were made to the registers or the PSW, and therefore need to be reflected in the program environment before the program resumes running. See [“MVS-related information” on page 644](#).

**Note:** The PT\_RECOVER request, along with PT\_TRACE\_ME and PT\_EXTENDED\_EVENT, is issued from the process that is to be debugged. All other ptrace service requests are issued from the debugger.

## Handling extended events in a debugged process

Language Environment supports a generic debugger interface for the high-level languages it supports, such as C. This interface requires that a module named CEEEVDBG be available for Language Environment to load and call when certain events occur in a high-level language program that has had this interface enabled via a TEST run time option. The input to CEEEVDBG is a parameter list that contains an event code and information that is associated with that code.

To allow events that are invoked in this manner to be used by a debugger using ptrace, z/OS UNIX ships a sample CEEEVDBG module. This module "glues" the Language Environment Interactive Debug Event Handler interface to the debugger, using the ptrace PT\_EXTENDED\_EVENT request to create an extended event. However, any product can choose to use the PT\_EXTENDED\_EVENT request to create extended events in this manner; CEEEVDBG is just an example of the proper usage. The relationship between the PT\_EXTENDED\_EVENT, PT\_EVENTS and PT\_EXPLAIN requests can best be illustrated with a usage scenario:

1. The debug tool interface is enabled for the program to be debugged. For more information, see [Debug event handler in z/OS Language Environment Vendor Interfaces](#).
2. The sample CEEEVDBG module is installed so that Language Environment can load it. More information is provided in [“MVS-related information” on page 644](#).
3. The debugger issues one or more PT\_EVENTS requests to establish the set of extended events for which it has an interest. This should normally be done just after the target program has been placed into debugging mode, during debugger initialization regarding the debugged program. There are, however, no restrictions on modifying the list of extended events any time the debugged program is stopped for an event.
4. The program is allowed to run. When Language Environment encounters certain events (for example, a mutex initialization, lock, or unlock), it invokes CEEEVDBG with the appropriate event code.
5. CEEEVDBG collects certain information about the event and issues the PT\_EXTENDED\_EVENT request to invoke ptrace. The information that is collected consists of the event code and registers 1, 12 and 13 at input to CEEEVDBG. Register 1 contains the address of the parameter list that contains the event code and associated information. Registers 12 and 13 contain the addresses of Language Environment control blocks that the debugger can use to gather additional information. The extended event information structure is defined in the BPXYPTRC macro.
6. The PT\_EXTENDED\_EVENT request filters the input event code with the set of events established with the PT\_EVENTS requests. If the input event code is found in the list, an Extended Event is initiated. This causes the debugged program to stop and the debugger to be notified. The corresponding wait() status reported to the debugger is WastStopFlagExtended.

7. The debugger reacts to the unique wait() status by issuing the PT\_EXPLAIN request. This request returns the information collected by the PT\_EXTENDED\_EVENT request to the debugger.
8. Because the information is in the form of addresses, the debugger must issue PT\_READ\_D or PT\_READ\_BLOCK requests to obtain the associated extended event information.

## Manipulating data in a debugged process

You can use the ptrace service to look at the following types of data in the debugged process; some might be altered.

1. **General or machine control registers.** This includes general-purpose registers (GPRs), floating-point registers (FPRs), vector registers (VR), control registers (CRs), and the program status word (PSW). Control registers can only be looked at, never modified. Control registers contain system information, and their content is not necessarily related to the debugged process. The value of some of the control registers might change from one call of PTRACE to the next, even when the debugged process is stopped across both calls. The entire PSW can be looked at, but only the rightmost 4 bytes (the instruction counter and addressing mode) can be changed. The PT\_READ\_GPR and PT\_WRITE\_GPR requests are used for all registers except the FPRs, and the interface supports 4 bytes only. As a result, the PSW must be accessed with two ptrace service requests, each specifying the register number for the appropriate half of the PSW.

The PT\_READ\_VR and PT\_WRITE\_VR requests are used for the vector registers, and this interface supports 16 bytes. All the register numbers are defined in the BPXYPTRC macro.

**Restriction:** Only the second fullword of the PSW can be written into.

The PT\_READ\_FPR and PT\_WRITE\_FPR requests are used for the FPRs, and this interface supports 8 bytes. In addition to reading and writing the floating point registers, you can also read and write the floating point control register. All the register numbers are defined in the BPXYPTRC macro.

Two special cases exist. One is the PT\_REGSET request, which returns all the general-purpose registers. The second is the PT\_CONTINUE request, which can indicate that the program should continue at a specified address. In other words, that the instruction counter should be modified.

2. **User program storage.** This takes two forms. The first is for fullword requests, which look at or modify a fullword of storage only. The PT\_READ\_I, PT\_READ\_D, PT\_READ\_U, PT\_WRITE\_I, and PT\_WRITE\_D requests are used to accomplish this. For MVS considerations, see [“MVS-related information” on page 644](#). The user area request (PT\_READ\_U) operates on the user area. For more information, see [“User area description” on page 642](#).

The second form is for blocks of storage, up to a defined maximum length. For this, the PT\_READ\_BLOCK and PT\_WRITE\_BLOCK requests are used. The maximum defined length is defined in the BPXYPTRC macro.

3. **Blocking requests.** Most of the requests described here can be blocked into a single ptrace call by using the PT\_BLOCKREQ request. This saves system resources when a large amount of information must be read or written. The PT\_BLOCKREQ request can be used, for example, to read or write all the GPRs, all the FPRs, and several areas of user program storage on a single request. The PtBRInfo structure, defined in macro BPXYPTRC, defines the mechanism for blocking several requests into a single request.

## Setting a breakpoint in a debugged process

You can use the PT\_WRITE\_I (or PT\_WRITE\_D or PT\_WRITE\_BLOCK) request to store SVC 144 instructions into a debugged program. The SVC 144 instruction causes an SVC 144 event to be recognized by the debugger. See [“MVS-related information” on page 644](#) for MVS considerations regarding the use of SVC 144.

**Note:** It is the responsibility of the caller to save and restore the actual program instructions that are overlaid by inserted breakpoint SVCs. You can use ptrace services to accomplish this, but no implicit understanding or management of program instructions is done by the ptrace service.

## Capturing storage in a debugged process.

When you use the standard ptrace requests previously discussed, you pay a performance penalty when you perform certain operations. For example, stepping over a breakpoint instruction while leaving the breakpoint in the program requires several ptrace requests, as in the following scenario:

- PT\_WRITE\_I to restore the original instruction over the SVC 144
- PT\_READ\_I to get the fullword following the restored instruction
- PT\_WRITE\_I to insert a temporary SVC 144 after the restored instruction
- PT\_WRITE\_GPR to back up the PSW to point to the restored instruction
- PT\_CONTINUE to execute the restored instruction and hit the temporary breakpoint
- PT\_WRITE\_I to restore the temporarily overlaid instruction
- PT\_WRITE\_I to reinsert the SVC 144 at its original location
- PT\_WRITE\_GPR to back up the PSW to point to the restored temporary instruction
- PT\_CONTINUE to resume running until the next event

Each of these ptrace requests consumes system resources and requires some amount of time to complete. The cumulative effect might be performance that is slower than expected.

The PT\_CAPTURE request allows you to capture one or more virtual pages of storage in the debugged process into a buffer in your address space. After capturing storage in this manner, you have shared write access to the storage, and can access it directly by accessing the returned buffer. This allows you to bypass those ptrace requests that would normally be used to read or write storage in the debugged process. One use for the PT\_CAPTURE request could be to capture the entire debugged program load module. Then, using the same example of stepping over a breakpoint instruction, you could eliminate all but the PT\_WRITE\_GPR and PT\_CONTINUE requests by directly placing SVC 144 instructions and restoring program instructions in the captured buffer. Any storage that is accessible by the debugged program can be captured in this manner.

Storage that is captured using the PT\_CAPTURE request is always on a 4K page boundary, and the minimum amount of storage captured is one 4K page. You are responsible for determining the correct offset of the desired storage in the captured buffer. For example, if the address you want to capture is 3A094BE8, the PT\_CAPTURE request captures the entire page starting at 3A094000. If the service returns a capture buffer address of 35081000, the start of the desired storage in this buffer is 35081BE8.

The PT\_UNCAPTURE request is used to free a specific buffer or all captured buffers. Freeing the buffer by using this request severs the capture relationship between the captured storage and the local buffer. To free a specific buffer, pass the buffer address on the ptrace request. To free all buffers, pass a 0 buffer address.

## Resuming or detaching from a debugged process.

To cause a stopped, debugged process to resume running, you use the ptrace PT\_CONTINUE service request. The request specifies whether running is to continue from where it was stopped, or at another instruction counter address. It also specifies whether the process is to continue as though no signal, or a specified signal, had just been received.

These two functions of the PT\_CONTINUE request can be used to accomplish several debugging objectives. For instance, if the debugged program was stopped by a particular signal (for instance, SIGINT), the debugger can indicate that the program can continue normally, and can continue as though a SIGINT had just arrived. In effect, this allows the program to continue as though it had not been interrupted by the ptrace service. The debugger could also choose to ignore the signal that stopped the process (again assume a SIGINT), by specifying PT\_CONTINUE without a signal. This allows the program to resume running, but the original SIGINT is discarded before it is delivered to the debugged program.

When a debugger finishes debugging a program, it uses the PT\_DETACH request to take the process out of debugging mode and allow it to continue. A signal can be supplied on this request, as it is with the PT\_CONTINUE request.

When a process is continued using these ptrace service requests, all signals that are pending on the focus thread, and all signals that are pending on the process (other than the ones supplied on the PT\_CONTINUE or PT\_DETACH service request, or SIGKILL), are discarded.

When a debugged process is stopped because of a signal, or is waiting for a signal to arrive when the PT\_CONTINUE or PT\_DETACH request is issued, there are special considerations:

- If no signal is supplied on the ptrace service request, the process continues running immediately. If it was in a stopped state, it behaves as if a SIGCONT had just arrived. If it was waiting for a signal, it behaves as if a signal had just arrived.
- If a signal is supplied on the ptrace service request, that signal takes whatever action it normally would with respect to the state of the debugged process.

For example, if the process was in a stopped state, and a SIGCONT is supplied on the request, the process is taken out of the stopped state. However, if it was also waiting for the arrival of a signal, it still waits after the ptrace service request. Likewise, a signal whose action is to wake up processes that are waiting for a signal does so if the debugged process was waiting for a signal. If the process was in a stopped state, however, it remains in a stopped state after the ptrace service request has been processed.

## Ending a debugged process

To end a stopped, debugged process, you can use the ptrace PT\_KILL service request. This causes the process to end as though it had received a SIGKILL signal. You can also use the PT\_CONTINUE request to continue with a signal whose action is to end the process, although this has the effect of ending the process with the specified signal instead of with a SIGKILL.

## Multiprocess debugging mode

Multiprocess debugging mode allows a debugger to control more than one process. The debugger uses the ptrace PT\_MULTI service request to turn multiprocess mode on or off for a target process.

When multiprocess mode is in effect, the behavior of the exec, fork, attach\_exec, attach\_execmvs, and spawn services is modified. For the exec service, the only change is that the Status\_field parameter on the wait service issued by the debugger indicates that the process stopped for the exec service, instead of that it was stopped by the SIGTRAP signal.

For the fork service, the Status\_field parameter on the debugger wait service indicates that the process stopped for the fork call. In addition, the fork service causes both the parent and the new child process to stop, and the debugger gets status for both processes with the wait service. The debugger should issue the wait service until it receives status for both the parent and child processes. This is different from multiprocess mode's not being in effect; in this case neither the parent nor child process stops because of the fork service, and the debugger is not made aware of the fork event at all.

For the attach\_exec and attach\_execmvs services, the Status\_field parameter on the debugger wait service indicates that the process stopped for a local fork. In addition, these services cause both the parent and the new child process to stop, and the debugger gets status for both processes with the wait service. The debugger should issue the wait service until it receives status for both the parent and child processes. This is different from multiprocess mode's not being in effect; in this case, neither the parent nor child process stops because of these services, and the debugger is not made aware of the local fork event at all. After the notification of the local fork event, the attach\_exec service loads the executable program into storage and causes the Status\_field parameter on the wait service issued by the debugger to indicate that the process stopped for the exec service, instead of that it was stopped by the SIGTRAP signal.

For the spawn service, the effects are a combination of those described for fork and attach\_exec. The parent presents status like that for a fork call (the debugger wait Status\_field indicates that the process stopped for a fork). The child presents status like that for attach\_exec (the debugger wait Status\_field indicates that the process stopped for a local fork).

## User area description

The PT\_READ\_U request is used with the user area for a target process. The user area is a collection of control information. It is not necessarily a contiguous storage area, and it is not readily accessible by an end user except via the PT\_READ\_U request.

To access the user area, an offset, as opposed to an absolute address, must be supplied on the PT\_READ\_U request. Each unit of control information is a fullword, and the offsets represent each multiple of 4 bytes. The offsets begin with 1 and progress by 1. Table 27 on page 642 shows the offsets and the associated control information that are defined in the BPXYPTRC macro (see [“BPXYPTRC — Map parameters for ptrace”](#) on page 1127):

Table 27. Offsets and associate information that are defined in BPXYPTRC	
Constant	Control information
<b>PtUArea#MinSig–PtUArea#MaxSig</b>	Signal catcher information for signal numbers 1 - 64 (the rest of the range is reserved). Not all potential signal numbers are valid; the valid signal numbers are defined in the BPXYSIGH macro. Signal catcher information is one of the following (the constants for signal default and ignore actions are defined in the BPXYSIGH macro): <ul style="list-style-type: none"> <li>• SIG_DFL#: Take default action for this signal</li> <li>• SIG_IGN#: Ignore this signal</li> <li>• Address: Address of the signal catcher function</li> </ul>
<b>PtUArea#IntCode</b>	Program interrupt code, in the following format: <ul style="list-style-type: none"> <li>• Bytes 0 and 1: unused</li> <li>• Bytes 2 and 3: program interrupt code in hexadecimal</li> </ul>
<b>PtUArea#AbendCC</b>	Abend completion code, in the following format: <ul style="list-style-type: none"> <li>• Byte 0: flags</li> <li>• Bytes 1–3: system completion code (first 12 bits) and user completion code (second 12 bits)</li> </ul>
<b>PtUArea#AbendRC</b>	Abend reason code
<b>PtUArea#SigCode</b>	Signal code, in the following format: <ul style="list-style-type: none"> <li>• Bytes 0 and 1: unused</li> <li>• Bytes 2 and 3: signal code in hex</li> </ul>
<b>PtUArea#ILC</b>	Instruction length code, in the following format: <ul style="list-style-type: none"> <li>• Bytes 0–2: unused</li> <li>• Byte 3: instruction length code</li> </ul>

The PT\_READ\_U request can therefore be used to obtain additional information about signals; or when a debugger is notified that a debugged process stopped with a SIGILL, SIGSEGV, SIGFPE, or SIGABND signal.

## Related services

- [“attach\\_exec \(BPX1ATX, BPX4ATX\) — Attach a z/OS UNIX program ”](#) on page 48
- [“attach\\_execmvs \(BPX1ATM, BPX4ATM\) — Attach an MVS program ”](#) on page 58
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program”](#) on page 163
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ”](#) on page 227



- [“loadhfs \(BPX1LOD, BPX4LOD\) — Load a program into storage by path name” on page 393](#)
- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)
- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)

## Characteristics and restrictions

The following restrictions apply to the use of the ptrace service:

1. The ptrace service is supported from programs that are running in PSW key 8 mode only. Calls to the ptrace service that are made from debugger programs (authorized or problem state) with other than key 8 are rejected with an error code.
2. A process that is being debugged must not be running if any of the following environmental conditions are true:
  - It is running in access register (AR) mode.
  - It is running in supervisor PSW state.
  - It is running with a PSW key not equal to 8.
  - It is running with APF authorization, and the debugger process does not have read permission to the BPX.DEBUG resource in the FACILITY class.
  - It is running with the security product function called Program Access to Data Support (PADS) activated.

A process that is running with any of these conditions ends abnormally if it attempts to use the ptrace service to notify the debugger of a ptrace service event.

3. A SIGKILL signal that is sent to a process that is being debugged by the ptrace service cannot be trapped. When a SIGKILL signal ends a process, the ptrace service is not given a chance to intervene.

**Note:** SIGKILL is delivered to the target process according to normal signal delivery rules. If the target process is stopped, but is not waiting for signals (for example, if it is stopped for a ptrace service event), the SIGKILL remains pending until the process resumes (using the same example, when a PT\_CONTINUE, PT\_DETACH, or PT\_KILL ends the ptrace service event).

4. The PT\_ATTACH, PT\_REATTACH, and PT\_REATTACH2 requests cannot be issued with a target process that is a system address space (see [“MVS-related information” on page 644](#)). If this attempt is made, the EPERM error is returned in the Return\_code parameter.
5. The PT\_ATTACH, PT\_REATTACH, and PT\_REATTACH2 requests cannot be issued with a target process that is the INIT process (with a process identifier equal to 1). If this attempt is made, the EPERM error is returned in the Return\_code parameter.
6. The PT\_ATTACH, PT\_REATTACH, and PT\_REATTACH2 requests cannot be issued with a target process that is the parent of the calling process. If this attempt is made, the ESRCH error is returned in the Return\_code parameter.
7. The PT\_ATTACH, PT\_REATTACH, and PT\_REATTACH2 requests cannot be issued with a target process that uses the setuid service to set the uid to 0, unless the process is also running with superuser equal to daemon authority (in other words, is running without an active BPX.DAEMON resource profile in the FACILITY class). If this attempt is made, the EPERM error is returned in the Return\_code parameter.
8. The debugger cannot use multiple threads within a single process to debug multiple target processes. If multiprocess debugging is desired, either a single thread debugger process must be associated with all debugged processes, or the debugger must use multiple processes, where the association with debugged processes is on a one-to-one basis.
9. The debugger should not have a signal catcher for the SIGCHLD signal. The ptrace service uses the SIGCHLD signal for internal communication, and the use of a catcher by the debugger would interfere with this communication. The most visible result of using a SIGCHLD catcher would be EINTR errors returned for most ptrace service requests, although other unpredictable results could also occur.

10. To ensure that pending signal information for the focus thread is not lost (because the signals were discarded), the PT\_THREAD\_INFO request should be the first ptrace request that is issued when the debugger gets notified of a ptrace service event.
11. The following requests are not supported while a debugged process is stopped for a local fork child event:
  - PT\_READ\_I
  - PT\_READ\_D
  - PT\_READ\_BLOCK
  - PT\_READ\_GPR
  - PT\_READ\_GPRH
  - PT\_READ\_FPR
  - PT\_WRITE\_I
  - PT\_WRITE\_D
  - PT\_WRITE\_BLOCK
  - PT\_WRITE\_GPR
  - PT\_WRITE\_GPRH
  - PT\_WRITE\_FPR
  - PT\_REGHSET
  - PT\_REGSET
  - PT\_CONTINUE to continue at a specified address
  - PT\_READ\_VR
  - PT\_WRITE\_VR
12. The following requests are not supported while a debugged process is stopped for an extended event:
  - PT\_READ\_GPR
  - PT\_READ\_GPRH
  - PT\_READ\_FPR
  - PT\_WRITE\_GPR
  - PT\_WRITE\_GPRH
  - PT\_WRITE\_FPR
  - PT\_REGHSET
  - PT\_REGSET
  - PT\_CONTINUE to continue at a specified address
  - PT\_READ\_VR
  - PT\_WRITE\_VR
13. If the debugger is running in a multi-thread process, then the SIGCHLD signal must be blocked on all threads except the one issuing the BPX1PTR call.

### Examples

See [“BPX1PTR \(ptrace\) example” on page 1274](#).

### MVS-related information

1. As a result of the PT\_LDINFO request, the ptrace service invokes the Contents Supervisor CSVINFO service. CSVINFO returns information about load modules in the debugged process based on CSV control blocks. This information is then returned to the caller of the PT\_LDINFO request. CSVINFO

uses the MVS macros ATTACH, LINK or XCTL; or the exec or loadhfs service to return information about all modules brought into storage by any task in the process.

2. PT\_READ\_GPR requests that read the machine control registers (CRs) can return CR information that is not consistent with the user program that is being debugged. This is because the ptrace service reads the actual hardware registers that probably have changed because of internal PC invocations.
3. No distinction is made between the instruction area (\_I) or data area (\_D) for the PT\_READ\_I, PT\_READ\_D, PT\_WRITE\_I, and PT\_WRITE\_D requests. These are all treated as user storage requests.
4. A debugger cannot set breakpoints in programs that are loaded into read-only storage (for example subpool 252 or LPA). Users of debugger programs that use ptrace must be aware of the storage location of their programs, and, if necessary, take appropriate steps to ensure that the programs are loaded into read/write storage (for example, subpool 251).
5. The PT\_RECOVER request can be issued by ESPIE and ESTAE routines that capture program checks in user programs. The main requirement is that any registers or PSW values that are changed by the debugger after it recognizes the program check event be restored before the user program resumes running. Also, if a signal is sent to the debugged program by the user recovery routine, it must be sent outside of the user recovery routine (ESPIE or ESTAE). This ensures that signal delivery operates in the correct environment.
6. SVC 144 instructions can be inserted only into storage key 8 user programs. You cannot use SVC 144 instructions to do breakpoint debugging of system (key 0) routines.

The SVC 144 routine has the following characteristics:

- SVC 144 is a type-3 SVC.
  - The user program registers and PSW that are saved by the SVC 144 routine are changed if requested by PT\_WRITE\_GPR requests.
  - Any modification that is made to register 14 with a PT\_WRITE\_GPR request is lost. This is because the SVC 144 routine uses register 14 to exit.
  - If the process under which the SVC 144 routine runs is not in ptrace mode (started with a PT\_TRACE\_ME, PT\_ATTACH, PT\_REATTACH or PT\_REATTACH2 request), the routine abends the caller.
  - If the SVC 144 routine is called while the process is in access register mode, supervisor state, or any key other than 8, the routine abends the caller. In addition, APF-authorized invocation is not allowed unless the debugger has read permission to the BPX.DEBUG resource in the FACILITY class.
7. MVS system address spaces cannot be debugged with the ptrace service. A system address space is identified by one of the following:
    - A command scheduling control block (CSCB) does not exist. The master address space is an example of an address space with no CSCB.
    - The CSCB identifies the address space as a system address space.
  8. The sample CEEEVDBG module must be installed as follows:
    - The sample CEEEVDBG module is in the form of source code that is written in basic assembler language. This module must be assembled with the following Language Environment macros made available to the assembler: CEECAA, CEEDSA, CEEENTRY, CEEPPA.
    - The object deck must be link-edited with the object deck for the ptrace system call stub.
    - The load module must be placed into a load library that is accessible by Language Environment.

# querydub (BPX1QDB, BPX4QDB) – Obtain the dub status of the current task

## Function

The querydub callable service obtains the dub status information for the current task. The status information indicates whether the current task has already been dubbed, is ready to be dubbed, or cannot be dubbed as a process (or thread).

## Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1QDB):	31-bit
AMODE (BPX4QDB):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1QDB,(Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4QDB with the same parameters.

## Parameters

**Return\_value**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the querydub service returns -1 if the request is unsuccessful. Otherwise it contains the returned status, which can have one of the following values:

Value	Description
QDB_DUBBED_FIRST	The task has already been dubbed. This task and this RB caused the dub.
QDB_DUBBED	The task has already been dubbed. Another task or another RB caused the dub.

Value	Description
<b>QDB_DUB_MAY_FAIL</b>	The task has not been dubbed; an attempt to dub the task may fail. The most likely reason for failure may be a missing or incomplete user security profile; or the lack of an OMVS segment.
<b>QDB_DUB_OKAY</b>	The task has not been dubbed; an attempt to dub the task will probably succeed. The service has determined that an OMVS segment exists for the task. However, it has not checked for other potential errors. It is possible that the task may not have the proper UID and GID set up in the security profile, causing a subsequent dub failure.
<b>QDB_DUB_AS_PROCESS</b>	The task has not been dubbed, but its address space has. An attempt to dub the task will cause the task to be dubbed as another process within the address space.
<b>QDB_DUB_AS_THREAD</b>	The task has not been dubbed, but its address space has. An attempt to dub the task will cause the task to be dubbed as a thread within the process (address space).

These constant values are defined in [“BPXYCONS — Constants used by services”](#) on page 1062.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the querydub service stores the return code. The querydub service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the querydub service stores the reason code. The querydub service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### Related services

None.

### Characteristics and restrictions

None.

### Examples

See [“BPX1QDB \(querydub\) example”](#) on page 1276.

## queue\_interrupt (BPX1SPB, BPX4SPB) – Return the last interrupt delivered

---

### Function

The queue\_interrupt callable service returns to the kernel the last interrupt that was delivered to the signal interface routine (SIR). The interrupt can be a signal, a cancelation request, or a quiesce request.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPXB1SPB):</b>	31-bit
<b>AMODE (BPX4SPB):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1SPB, (Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SPB with the same parameters.

### Parameters

#### Return\_value

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword to which the queue\_interrupt service returns 0 if it has permission to return the specified interrupt for delivery at the next kernel call. If no interrupt is returned, -1 is returned.

#### Return\_code

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword in which the queue\_interrupt service stores the return code. The queue\_interrupt service returns Return\_code only if Return\_value is -1. For a list of return

code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The queue\_interrupt service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value of Signal in the PPSD at the time this service was invoked was an unsupported signal. Either there was a storage overlay in the PPSD, or no signal was ever delivered to this task.
<b>EPERM</b>	The caller does not have permission to return the interrupt now. All signals must be blocked, and the task must invoke mvssigsetup before the queue_interrupt service is invoked. The following reason codes can accompany the return code: JRSignalsNotBlocked and JRNotSigsetup.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the queue\_interrupt service stores the reason code. The queue\_interrupt service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The data that is mapped by the BPXYPPSD control block is used by the queue\_interrupt service, and therefore should not be modified by the invoker, because this may result in an EINVAL return code.
2. The queue\_interrupt returns the interrupt to the kernel and restores the signal blocking mask to its preinterrupt state. The signal is acted on at the end of the next service.
3. When the PPSDJUMPBACK flag is set on in the BPXYPPSD (see “BPXYPPSD — Map signal delivery data” on page 1124) and the queue\_interrupt call is valid, control is not returned to the instruction after the queue\_interrupt invocation. Instead, it is returned to the point of the signal interruption that was just queued back to the kernel. General and access registers are restored to the values saved in the PPSD at the time of the interrupt.
4. When the PPSDREDRIVE flag is set on in the BPXYPPSD, the kernel is responsible for rescheduling the queued signal to interrupt the current thread at a later time. The signal interface routine (SIR) is no longer responsible for issuing another syscall to cause delivery of the signal. In fact, nonblocking syscalls (syscalls that do not return EINTR) do not cause delivery of pending deliverable signals when a redrive signal is in progress. Delivery of the signal only occurs when the redrive time limit expires. This time limit is maintained by the kernel and cannot be specified by the user.
5. When the PPSDMASKONLY flag is set on, the kernel does not requeue the signal; it only resets the signal mask to the value in PPSCURRENTMASK.

## Related services

- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)
- [“pthread\\_quiesce \(BPX1PTQ, BPX4PTQ\) — Quiesce threads in a process” on page 598](#)
- [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#)

## Characteristics and restrictions

The intended use of the queue\_interrupt service is from the signal interface routine that is specified on [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#). Although the queue\_interrupt service can be used anywhere, all signals must be blocked, and the task must set up signals by invoking

the mvssigsetup service before calling queue\_interrupt. See [Appendix G, “The relationship of z/OS UNIX signals to callable services,”](#) on page 1413.

### Examples

See [“BPX1SPB \(queue\\_interrupt\) example”](#) on page 1292.

## quiesce (BPX1QSE, BPX4QSE) – Quiesce a file system

---

### Function

The quiesce callable service quiesces a file system, making the files in it unavailable for use. After the file system is quiesced, the system can back up the data in it.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1QSE):</b>	31-bit
<b>AMODE (BPX4QSE):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Serialization:</b>	Enabled for interrupts
<b>Locks:</b>	No locks held
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1QSE,(File_system_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QSE with the same parameters.

### Parameters

#### File\_system\_name

Supplied parameter

#### Type:

Character string

#### Character set:

Printable characters

#### Length:

44 bytes

The name of a 44-character field that contains the file system name. The name must be left-justified and padded with blanks.



**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the quiesce service returns one of two values:

- 0, if the request is successful and the file system was mounted at the time of the request.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the quiesce service stores the return code. The quiesce service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The quiesce service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	The resource was temporarily unavailable. The following reason code can accompany the return code: JRLockErr.
<b>EBUSY</b>	The file system that was specified is being unmounted or has already been quiesced; or there are no more locks available. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced, and JRUnmountInProgress.
<b>EINVAL</b>	The file system that was specified cannot be quiesced. The following reason code can accompany the return code: JRInvalidParms.
<b>ENODEV</b>	The file system that was specified is not mounted. The following reason code can accompany the return code: JrFileSysNotThere.
<b>EPERM</b>	The user cannot request this service because it lacks the required permission. The following reason code can accompany the return code: JRUserNotPrivileged.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the quiesce service stores the reason code. The quiesce service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. After a quiesce service request, the file system is unavailable for use until a subsequent unquiesce service request is received.

2. Users accessing files in a quiesced file system are suspended until an unquiesce request for the file system is processed.

## Related services

- [“unquiesce \(BPX1UQS, BPX4UQS\) — Unquiesce a file system ” on page 975](#)

## Characteristics and restrictions

To quiesce a file system, the requester must be a superuser or, at least, have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class. This is the same authority that is required for mounting or unmounting a file system.

When a system joins the sysplex and processes mounts during initialization, any file system mounted in the sysplex that is in a quiesced state will not be mounted on that system at that time. When the quiesced file system is unquiesced, that file system will be mounted on any systems in the sysplex that do not have it already mounted.

## Examples

See [“BPX1QSE \(quiesce\) example” on page 1277](#).

# read (BPX1RED, BPX4RED) — Read from a file or socket

---

## Function

The read callable service reads the number of bytes that you specify from a file or socket into a buffer that you provide.

## Requirements

### Operation

### Environment

### Authorization

Supervisor state or problem state, any PSW key

### Dispatchable unit mode

Task

SRB - AF\_INET/AF\_INET6 socket support only

### Cross memory mode

PASN = HASN

### AMODE (BPX1RED)

31-bit task or srb mode

### AMODE (BPX4RED)

64-bit task mode only

### ASC mode

Primary mode

### Interrupt status

Enabled for interrupts

### Locks:

Unlocked

### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RED,(File_descriptor,
              Buffer_address,
              Buffer_ALET,
              Read_count,
              Return_value,
```

```
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4RED with the same parameters. The Buffer\_address parameter is a doubleword.

## Parameters

### File\_descriptor

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the file descriptor of an open file or socket.

### Buffer\_address

Parameter supplied and returned

#### Type

Address

#### Length

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer into which data is to be read.

### Buffer\_ALET

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the ALET for the Buffer\_address that identifies the address space or data space where the buffer resides.

You should specify a Buffer\_ALET of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### Read\_count

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the number of bytes that you want to read from the file. This number must be less than or equal to the length of the buffer that you provide for data to be read into.

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the read service returns the number of bytes that were actually read (this may be 0) if the request is successful, or -1 if it is not successful.

For more information about the return value, refer to usage note [“5” on page 655](#).

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the read service stores the return code. The read service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The read service can return one of the following values in the Return\_code parameter:

<b>Return code</b>	<b>Explanation</b>
<b>EAGAIN</b>	The file was opened with the nonblock option, and data is not available to be read.
<b>EBADF</b>	The File_descriptor parameter does not contain the descriptor of an open file; or the file is not opened for read. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINTR</b>	The service was interrupted by a signal before it could read any data.
<b>EINVAL</b>	The Read_Count parameter contains a value that is less than zero; or the socket is marked shutdown for read. The following reason codes can accompany the return code: JRSocketClosed, JRSocketCallParmError.
<b>EIO</b>	The process is in a background process group, and is attempting to read from its controlling terminal. Either the process is ignoring or blocking the SIGTTIN signal, or the process group is orphaned.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutOfSocketCells.
<b>ENOTSOCK</b>	The Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EWOULDBLOCK</b>	<ul style="list-style-type: none"> <li>• The socket is marked nonblocking, and no data is waiting to be received. The following reason code can accompany the return code: JRWouldBlock.</li> <li>• The socket is marked blocking, and the call has blocked, without receiving any data, for the time period specified in the SO_RCVTIMEO option. The following reason code can accompany the return code: JRTimeout. <ul style="list-style-type: none"> <li>– If AT-TLS is being used to provide transparent TLS/SSL support for a TCP socket and EWOULDBLOCK is returned on a blocking socket, reissue the select call or receive call to continue. For more information about AT-TLS and determining whether a TCP connection is using AT-TLS, see <a href="#">Application Transparent Transport Layer Security (AT-TLS)</a> in <i>z/OS Communications Server: IP Programmer's Guide and Reference</i>.</li> </ul> </li> </ul>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the read service stores the reason code. The read service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. See [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041 for more information about programming considerations for SRB mode.
2. Read\_count: The value of Read\_count is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.
3. Access time: A successful read updates the access time of the file read.
4. Origin of bytes read: If the file that is specified by File\_descriptor is a regular file, or any other type of file where a seek operation is possible, bytes are read from the file offset that is associated with the file descriptor. A successful read increments the file offset by the number of bytes that are read. For files for which no seek operation is possible, there is no file offset associated with the file descriptor. Reading begins at the current position in the file.
5. Number of bytes read: When a read request completes, the Return\_value field shows the number of bytes that were actually read—a number less than or equal to the number that was specified as Read\_count. The following are some reasons why the number of bytes read might be less than the number of bytes requested:
  - Fewer than the requested number of bytes remained in the file; the end of file was reached before Read\_count bytes were read.
  - The service was interrupted by a signal after some, but not all, of the requested bytes were read. (If no bytes were read, the return value is set to -1 and an error is reported.)
  - The file is a pipe, FIFO, or special file, and fewer bytes than Read\_count specified were available for reading.

There are several reasons why a read request might complete successfully with no bytes read. That is, with Return\_value set to 0. For example, zero bytes are read in these cases:

- The service specified a Read\_count of zero.
  - The starting position for the read was at or beyond the end of the file.
  - The file that is being read is a FIFO file or a pipe, and no process has the pipe open for writing.
  - The file that is being read is a subsidiary pseudoterminal, and a zero-length canonical line was written to the manager pseudoterminal.
  - A directory is being read and the Physical File System does not support simple reads from directories. `opendir()` and `readdir()` should be used.
6. Nonblocking: If a process has a pipe open for reading with nonblocking specified, a request to read from the file ends with a return value of -1 and a "Resource temporarily unavailable" return code. But if nonblocking is not specified, the read request is blocked (does not return) until some data is written, or until the pipe is closed by all other processes that have the pipe open for writing.
- Manager and subsidiary pseudoterminals also operate this way, except that how they act depends on how they were opened. If the manager or the subsidiary is opened blocking, the reads are blocked if there is no data. If they are opened nonblocking, EAGAIN is returned if there is no data.
7. SIGTTIN processing: The read service causes signal SIGTTIN to be sent under the following conditions:
    - The process is attempting to read from its controlling terminal, and
    - The process is running in a background process group, and
    - The SIGTTIN signal is not blocked or ignored, and
    - The process group of the process is not orphaned.

If these conditions are met, SIGTTIN is sent. If SIGTTIN has a handler, the handler gets control, and the read ends with the return code set to EINTR. If SIGTTIN is set to default, the process stops in the read and continues when the process is moved to the foreground.

8. If automatic conversion is enabled with Unicode Services (ALL), see [Using Unicode Services in z/OS UNIX System Services Planning](#) for more information.

## Related services

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset ” on page 406](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe” on page 551](#)

**Note:** The read service is not related to the **read** shell command.

## Characteristics and restrictions

If the file was opened by an authorized program, all subsequent reads and writes against the file must be issued from an authorized state.

The read (BPX1RED, BPX4RED) and write (BPX1WRT, BPX4WRT) callable services do not support simultaneous reading or writing of the same shared open file by different threads when one or both of the following are true:

1. Automatic conversion is enabled using Enhanced ASCII (ON) and different character set IDs (CCSIDs) are used.
2. Automatic conversion is enabled using Unicode Services (ALL) and different CCSIDs are used, or mixing read and write operations of multibyte characters are performed which result is storing of partial characters.

The first restriction is not applicable if each thread coordinates its reads and writes so that simultaneous I/O does not occur. Both restrictions are not applicable if each thread opens the file independently.

Reads or writes that cause a conversion of greater than 2 G result in an EINVAL error with reason JrUniOpTooBig.

Refer to the BPX1FCT (fcntl) operations for instances of how a lseek operation in a conversion environment can affect read and write operations

## Examples

See [“BPX1RED \(read\) example” on page 1280](#).

# readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory

## Function

The readdir callable service reads multiple name entries from a directory.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RDD)</b>	31-bit

<b>Operation</b>	<b>Environment</b>
<b>AMODE (BPX4RDD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RDD,(Directory_file_descriptor,
               Buffer_address,
               Buffer_ALET,
               Buffer_length,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RDD with the same parameters. The Buffer\_address parameter is a doubleword.

## Parameters

### Directory\_file\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened. For more information, see [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory” on page 523](#).

### Buffer\_address

Parameter supplied and returned

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer in which readdir is to write the directory entries. This address must be supplied to the readdir call. The directory entries are mapped by the BPXYDIRE macro; see [“BPXYDIRE — Map directory entries for readdir” on page 1074](#).

### Buffer\_ALET

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the ALET for the Buffer\_address that identifies the address space or data space where the buffer resides.

You should specify a Buffer\_ALET of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

**Buffer\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer\_address.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readdir service returns the number of directory entries that have been read into the buffer, or -1 if it is unsuccessful. A value of 0 in Return\_value indicates the end of the directory.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readdir service stores the return code. The readdir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The readdir service can return one of the following values in the Return\_code parameter:

<b>Return code</b>	<b>Explanation</b>
<b>EBADF</b>	The Directory_file_descriptor argument does not refer to an open directory.
<b>EINVAL</b>	The buffer is too small to contain any entries.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readdir service stores the reason code. The readdir service returns Reason\_code only if Return\_value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. This interface differs from the POSIX C high-level-language interface in that it returns more than one directory entry; it also returns the entries in the caller's buffer.



2. The buffer contains a variable number of variable-length directory entries. Only full entries are placed in the buffer, up to the buffer size specified, and the number of entries is returned.
3. Each directory entry that is returned has the following format (as shown on [“BPXYDIRE — Map directory entries for readdir”](#) on page 1074):
  - 2-byte Entry\_length. The total entry length, including itself.
  - 2-byte Name\_length. The length of the following Member\_name subfield.
  - Member\_name. A character field of length Name\_length. This name is not terminated by a null character.
  - File system-specific data. If Name\_length + 4 = Entry\_length, this subfield is not present.

The entries are packed together, and the length fields are not aligned on any particular boundary.
4. The buffer that is returned by one call to the readdir service must be used again on the next call to the readdir service, to continue reading entries from where you left off. The buffer must not be altered between calls, unless the directory has been rewound.
5. If the contents of the directory have changed (files have been added or removed) since a previous call to the readdir service, a call should be made to the rewinddir service so that the updated contents of the directory can be read.
6. The end of the directory is indicated in one of two ways:
  - A Return\_value of 0 entries is returned.
  - Some physical file systems may return a null name entry as the last entry in the caller's buffer. A null name entry has an Entry\_length of 4 and a Name\_length of 0.

The caller of the readdir service should check for both conditions.
7. zFS and TFS follow the UNIX standard and do not return names in sorted order.

## Related services

- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory”](#) on page 124
- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory”](#) on page 523
- [“rewinddir \(BPX1RWD, BPX4RWD\) — Reposition a directory stream to the beginning ”](#) on page 701

## Characteristics and restrictions

None.

## Examples

See [“BPX1RDD \(readdir\) example”](#) on page 1278.

# readdir2 (BPX1RD2, BPX4RD2) — Read an entry from a directory

## Function

The readdir2 callable service reads multiple name entries from a directory.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN

Operation	Environment
<b>AMODE (BPX1RD2):</b>	31-bit
<b>AMODE (BPX4RD2):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RD2,(Directory_file_descriptor,
              UIO,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RD2 with the same parameters. Some of the addresses in the UIO structure are doublewords.

## Parameters

### Directory\_file\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened (see [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory”](#) on page 523).

### UIO

Supplied and returned parameter

#### Type:

Structure

#### Length:

Fuio#Len (from the BPXYFUIO macro)

The name of an area that contains the user input and output block. This area is mapped by the BPXYFUIO macro (see [“BPXYFUIO — Map file system user I/O block”](#) on page 1078).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the readdir2 service returns the number of directory entries that have been read into the buffer that is pointed to by the UIO, or -1 if the request is unsuccessful. A value of 0 in Return\_value indicates the end of the directory.

### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readdir2 service stores the return code. The readdir2 service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The readdir2 service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EACCES</b>	The FuioChkAcc bit was set to request that an access check be performed, but the calling process does not have permission to read the specified directory.
<b>EBADF</b>	The Directory_file_descriptor argument does not refer to an open directory.
<b>EINVAL</b>	There was a parameter error; for example, a supplied area was too small. The following reason codes can accompany the return code: JRInvalidFuio, JrBytes2RWZero, JRRddPlusNoCursorSupp

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readdir2 service stores the reason code. The readdir2 service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. This interface differs from the POSIX C high-level-language interface in that it returns more than one directory entry, and it also returns the entries in the caller's buffer.
2. The buffer contains a variable number of variable-length directory entries. Only full entries are placed in the buffer, up to the buffer size specified, and the number of entries is returned.
3. Each directory entry that is returned has the following format, which is mapped by BPXYDIRE (see ["BPXYDIRE — Map directory entries for readdir"](#) on page 1074):

- 2-byte Entry\_length. The total entry length, including itself.
- 2-byte Name\_length. The length of the following Member\_name subfield.
- Member\_name. A character field of length Name\_length. This name is not terminated by a null character.
- File system specific data. If Name\_length + 4 = Entry\_length, this subfield is not present.

The entries are packed together, and the length fields are not aligned on any particular boundary.

4. The end of the directory is indicated when a Return\_value of 0 entries is returned.

In addition, some physical file systems may return a null name entry as the last entry in the caller's buffer. A null name entry has an Entry\_length of 4 and a Name\_length of 0. The caller of the readdir2 service should check for both conditions.

5. Two protocols are supported for reading through large directories with successive calls:

**Cursor protocol**

The cursor, or offset, that is returned in the UIO by the readdir2 service contains file-system-specific information that locates the next directory entry. The cursor and buffer must be

preserved by the caller from one readdir2 call to the next, and reading proceeds based on the cursor. The buffer must not be altered between calls, unless the directory has been rewound.

### **Index protocol**

The index that is set in the UIO by the caller determines which entry to start reading from. To read through the directory, the caller increments the index by the number of entries that were returned on the previous call.

Because this index represents the number of entries into the directory, the caller should be aware that if entries are being added or deleted in the directory while the call is being done, duplicate or missing entries could result.

The cursor protocol is preferred for better performance.

6. The cursor information that is returned from a call to readdir2() can be used on successive calls to readdir().
7. If the contents of the directory have changed (files have been added or removed) since a previous call to the readdir2 service, a call should be made to the rewinddir service so that the updated contents of the directory can be read.
8. The following UIO fields should be set to specify the details of the read directory request:

#### **FuioID**

Contains Fuio#ID (from the BPXYFUIO macro).

#### **FuioLen**

Contains the length of the UIO structure.

#### **FuioChkAcc**

Requests that the PFS perform required access checking before performing the requested readdir2 operation.

#### **FuioBufferAddr**

Contains the address of a buffer where the directory entries are to be returned.

#### **FuioBufferAlet**

Contains the ALET of the buffer where the directory entries are to be returned.

#### **FuioIBytesRW**

Specifies the maximum number of bytes that can be written to the output buffer.

#### **FuioRDIndex**

Specifies the first directory entry that is to be returned when the index protocol is used. The directory can be thought of as a 1-based array, and the index specifies which entry in the directory to begin reading from. When the FuioRDIndex is set to any nonzero value it will override any value in the FuioCursor field. To begin reading at the first directory entry, set the FuioRDIndex to 1.

#### **FuioCursor**

When the cursor protocol is used, this specifies a value, returned on the previous readdir2 call, that indicates the next entry to be read; or 0 on the first call. The FuioRDIndex must be set to 0 when the cursor protocol is being used. To begin reading at the first directory entry, both the FuioRDIndex and the FuioCursor should be set to 0.

#### **FuioRddPlus**

Indicates that the request is for the ReaddirPlus function. The attributes for each entry should be included in the output. If FuioRddPlus is specified then the Index protocol is used.

9. Some addresses in the UIO structure are doublewords, and some are not. If the buffer address is a 64-bit address, the caller must set the FUIOADDR64 flag in BPXYFUIO, and the FUIOBUFF64VADDR must contain the 64-bit virtual buffer address. When FUIOADDR64 is not set, the FUIOBUFFERADDR must contain the 31-bit virtual buffer address.
10. The following UIO fields are returned by the readdir2 service:

#### **FuioPSWKey**

This field is set to the caller's key.

**FuioCursor**

This field is set to the current cursor position after the readdir2 has occurred.

**FuioAsid**

This field is set to the caller's ASID.

**FuioCVerRet**

This field indicates that the Cookie Verifier (FuioCVer) is being returned.

**FuioCVer**

When FuioCVerRet is on, this field is set to the Cookie Verifier for the directory that is being read. When a directory is being read with multiple reads, you can use the FuioCVer that is returned to compare each Cookie Verifier with the previous one. If the directory has been modified between reads, you can reject the request, because the results will not be valid.

11. The buffer contents that are returned by the readdir2 service are mapped by the BPXYDIRE macro. See [“BPXYDIRE — Map directory entries for readdir”](#) on page 1074.

**Related services**

- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory”](#) on page 124
- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory”](#) on page 523
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory ”](#) on page 656
- [“rewinddir \(BPX1RWD, BPX4RWD\) — Reposition a directory stream to the beginning ”](#) on page 701

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1RD2 \(readdir2\) example”](#) on page 1279.

## read\_extlink (BPX1RDX, BPX4RDX) — Read an external symbolic link

---

**Function**

The read\_extlink callable service reads the contents of an external symbolic link into a buffer that you provide. The external symbolic link contains the external name that was specified when the symbolic link was defined (see [“extlink\\_np \(BPX1EXT, BPX4EXT\) — Create an external symbolic link ”](#) on page 183).

**Requirements**

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RDX)</b>	31-bit
<b>AMODE (BPX4RDX)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1RDX, (Link_name_length,
               Link_name,
               Buffer_length,
               Buffer_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RDX with the same parameters. The Buffer\_address parameter is a doubleword.

**Parameters****Link\_name\_length,**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of Link\_name.

**Link\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Link\_name\_length parameter

The name of a field that contains the link name of the external symbolic link that is to be read. The length of this field is specified in Link\_name\_length.

**Buffer\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer\_address.

**Buffer\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer that is supplied to the read\_extlink service, into which the value of the external symbolic link is to be written. The value of the external symbolic link is actually the external name that was specified when the symbolic link was created. The buffer must reside in the process's address space.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the read\_extlink service returns a count of the number of characters placed in the buffer, if the request is successful; or -1, if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the read\_extlink service stores the return code. The read\_extlink service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The read\_extlink service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	Search permission is denied for a component of the path prefix.
<b>EINVAL</b>	The file that is named by Link_name is not a symbolic link; or there was a problem with the supplied buffer. The following reason codes can accompany the return code: JRFileNotSymLink, and JRRdlBuffLenInvalid.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
<b>ENAMETOOLONG</b>	Link_name is longer than 1023 characters; or some component of the link name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file with the name specified by Link_name was found. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	A component of the path prefix is not a directory.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the read\_extlink service stores the reason code. The read\_extlink service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. If the buffer that is supplied to the read\_extlink service is too small to contain the value of the external symbolic link, the value is truncated to the length of the buffer (Buffer\_length). If the value that is returned is the length of the buffer, you can use the lstat service (see [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)) to determine the actual length of the external symbolic link.
2. If the Buffer\_length is 0, the value that is returned is the number of bytes in the external symbolic link. The buffer remains unchanged.
3. It is recommended that this function, rather than the readlink function (see [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link ” on page 666](#)), be used for reading an external link with a symbolic link ending its pathname.

## Related services

- [“extlink\\_np \(BPX1EXT, BPX4EXT\) — Create an external symbolic link ” on page 183](#)
- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)
- [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link ” on page 666](#)
- [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1RDX \(read extlink\) example” on page 1279](#).

# readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link

## Function

The readlink callable service reads the contents of a symbolic link into a buffer that you provide. The symbolic link contains the path name that was specified when the symbolic link was defined (see [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)).

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RDL)</b>	31-bit
<b>AMODE (BPX4RDL)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked



**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1RDL, (Link_name_length,
               Link_name,
               Buffer_length,
               Buffer_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RDL with the same parameters. The Buffer\_address parameter is a doubleword.

**Parameters****Link\_name\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of Link\_name.

**Link\_name**

Supplied parameter.

**Type**

Character string

**Character set**

No restriction

**Length**

Specified by the Link\_name\_length parameter

The name of a field that contains the link name of the symbolic link that is to be read. The length of this field is specified in Link\_name\_length.

**Buffer\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer\_address.

**Buffer\_address**

Supplied parameter.

**Type**

Address

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer that is supplied to readlink, into which the value of the symbolic link is to be written. The value of the symbolic link is the path name that was specified when the symbolic link was created. The buffer must reside in the process's address space.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the readlink service returns a count of the number of characters that are placed in the buffer if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the readlink service stores the return code. The readlink service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The readlink service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	Search permission is denied for a component of the path prefix.
<b>EINVAL</b>	The file named by Link_name is not a symbolic link or there was a problem with the supplied buffer. The following reason codes can accompany the return code: JRFileNotSymLink, JRRdlBuffLenInvalid.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
<b>ENAMETOOLONG</b>	Link_name is longer than 1023 characters; or some component of the link name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file with the name specified by Link_name was found. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	A component of the path prefix is not a directory.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the readlink service stores the reason code. The readlink service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. If the buffer that is supplied to the readlink service is too small to contain the value of the symbolic link, the value is truncated to the length of the buffer (Buffer\_length). If the value that is returned is the length of the buffer, you can use the lstat service (see [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name”](#) on page 409) to determine the actual length of the symbolic link.
2. If the Buffer\_length is 0, the value that is returned is the number of bytes in the symbolic link. The buffer remains unchanged.

## Related services

- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name”](#) on page 409
- [“readlinkat \(BPX1RLA, BPX4RLA\) — Read the value of a symbolic link relative to a directory file descriptor”](#) on page 669
- [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ”](#) on page 900
- [“symlinkat \(BPX1SYA, BPX4SYA\) — Create a symbolic link relative to a directory file descriptor ”](#) on page 905
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ”](#) on page 966
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor”](#) on page 969

## Characteristics and restrictions

None.

## Examples

See [“BPX1RDL \(readlink\) example”](#) on page 1278.

# readlinkat (BPX1RLA, BPX4RLA) — Read the value of a symbolic link relative to a directory file descriptor

---

## Function

The readlinkat callable service operates in the same way as the readlink callable service, except for the following differences:

- If the path name given in Link\_name is relative, then it is interpreted relative to the directory referred to by the file descriptor dirfd (rather than relative to the current working directory of the calling process, as is done by readlink() for a relative path name).
- If Link\_name is relative and dirfd is the special value AT\_FDCWD, then it is interpreted relative to the current working directory of the calling process, like readlink(). If Link\_name is absolute, then dirfd is ignored.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RLA)</b>	31-bit

Operation	Environment
<b>AMODE (BPX4RLA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RLA, (Link_name_length,
               Link_name,
               Buffer_length,
               Buffer_address,
               Dirfd,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RLA with the same parameters. All parameter addresses and addresses in parameter structure are doublewords.

## Parameters

### Link\_name\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of Link\_name.

### Link\_name

Supplied parameter.

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Link\_name\_length parameter

The name of a field that contains the link name of the symbolic link that is to be read. The length of this field is specified in Link\_name\_length.

### Buffer\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer\_address.

**Buffer\_address**

Supplied parameter.

**Type**

Address

**Length**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer that is supplied to readlinkat, into which the value of the symbolic link is to be written. The value of the symbolic link is the path name that was specified when the symbolic link was created. The buffer must reside in the process's address space.

**Dirfd**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains a file descriptor for the starting directory to be used for the make node request. The special value AT\_FDCWD that was specified with a relative path name, indicates to use the current working directory to interpret the path. Otherwise, a relative path name is interpreted relative to the directory referred to by this file descriptor. If path name is absolute, this parameter is ignored.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the readlinkat service returns a count of the number of characters that are placed in the buffer if the request is successful or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the readlinkat service stores the return code. The readlinkat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The readlinkat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Search permission is denied for a component of the path prefix.
<b>EBADF</b>	Dirfd is not a valid file descriptor or AT_FDCWD
<b>EINVAL</b>	The file named by Link_name is not a symbolic link or there was a problem with the supplied buffer. The following reason codes can accompany the return code: JRFileNotSymLink, JRRd1BuffLenInvalid.

Return_code	Explanation
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
<b>ENAMETOOLONG</b>	Link_name is longer than 1023 characters; or some component of the link name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file with the name specified by Link_name was found. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Not a directory returned for the following: <ul style="list-style-type: none"> <li>• Some component of Link_name is not a directory (JrCompNotDir).</li> <li>• The Dirfd parameter does not refer to a directory (JrNotDirFd).</li> </ul>

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the readlinkat service stores the reason code. The readlinkat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the buffer that is supplied to the readlinkat service is too small to contain the value of the symbolic link, the value is truncated to the length of the buffer (Buffer\_length). If the value that is returned is the length of the buffer, you can use the lstat service (see [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name”](#) on page 409) to determine the actual length of the symbolic link.
2. If the Buffer\_length is 0, the value that is returned is the number of bytes in the symbolic link. The buffer remains unchanged.

**Related services**

- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name”](#) on page 409
- [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link ”](#) on page 666
- [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ”](#) on page 900
- [“symlinkat \(BPX1SYA, BPX4SYA\) — Create a symbolic link relative to a directory file descriptor ”](#) on page 905
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ”](#) on page 966
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor”](#) on page 969

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1RLA \(readlinkat\) example”](#) on page 1281 and [“BPX4RLA \(readlinkat\) example”](#) on page 1373.

## readv (BPX1RDV, BPX4RDV) – Read data and store it in a set of buffers

---

### Function

The readv callable service reads data and stores it in a set of buffers.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
	SRB - AF_INET/AF_INET6 socket support only
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RDV)</b>	31-bit task or SRB mode
<b>AMODE (BPX4RDV)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1RDV,(File_descriptor,
               Iov_count,
               Iov_struct,
               Iov_alet,
               Iov_buffer_alet,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RDV with the same parameters. All addresses in parameter structures are doublewords.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor for which the readv request is to be done.

#### Iov\_count

Supplied and returned parameter

##### Type:

Integer

**Length:**

Fullword

The name of a field that contains the number of buffers that are pointed to by Iov\_struct. The total number of buffers may not exceed IOV\_MAX (defined in [“BPXYIOV — Map the I/O vector structure”](#) on page 1098).

**Iov\_struct**

Supplied parameter

**Type:**

Structure

**Length:**

Iov\_count times length(iov)

The name of a field that contains 31(64)-bit pointers to buffers in which data is to be stored, and their lengths. In 64-bit mode, Iov\_struct contains doubleword pointer and length subfields. See [“BPXYIOV — Map the I/O vector structure”](#) on page 1098 for more information.

**Iov\_alet**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the ALET for Iov\_struct.

**Iov\_buffer\_alet**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the ALET for buffers that are pointed to by Iov\_struct.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readv service returns one of the following:

- The number of bytes that were read into the buffers, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readv service stores the return code. The readv service returns Return\_code only if Return\_value is -1. For a list of return code values, see *Return codes (errno)* in *z/OS UNIX System Services Messages and Codes*. The readv service can return one of the following values in the Return\_code parameter:



Return code	Explanation
<b>EAGAIN</b>	The file was opened with the nonblock option, and data is not available to be read.
<b>EBADF</b>	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINTR</b>	A signal interrupted the readv function before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	The socket is marked shutdown for read; or an incorrect length was specified in the iov. The following reason codes can accompany the return code: JRSocketClosed, JRSocketCallParmError.
<b>EIO</b>	The process is in a background process group, and is attempting to read from its controlling terminal. However, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU signals, and the process group of the process is orphaned. This can happen, for example, if a background job tries to write to the terminal after the user has logged off.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>EWOULDBLOCK</b>	<ul style="list-style-type: none"> <li>• The socket is marked nonblocking and no data is waiting to be read, or the SO_RCVTIMEO timeout value was reached before data was available.</li> <li>• The socket is marked blocking, and the call has blocked for that time period that was specified in the SO_RCVTIMEO option without receiving any data. <ul style="list-style-type: none"> <li>– If AT-TLS is being used to provide transparent TLS/SSL support for a TCP socket and EWOULDBLOCK is returned on a blocking socket, reissue the select call or receive call to continue. For more information about AT-TLS and determining whether a TCP connection is using AT-TLS, see <a href="#">Application Transparent Transport Layer Security (AT-TLS)</a> in <i>z/OS Communications Server: IP Programmer's Guide and Reference</i>.</li> </ul> </li> </ul> <p>The following reason code can accompany the return code: JRTimeout.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the readv service stores the reason code. The readv service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

- See [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041 for more information about programming considerations for SRB mode.
- **Socket files:** When this callable service is used for datagram sockets, the readv service returns the entire datagram that was sent, providing that the datagram fits into the specified buffers. The excess is discarded. For stream sockets, data is not discarded. Multiple invocations of readv may be needed to return all the data.
- **Bytes read:** The number of bytes that were requested for reading is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.

- **Access time:** A successful read updates the access time of the file read.
- **Origin of bytes read:** If the file that was specified by File\_descriptor is a regular file, or any other type of file where a seek operation is possible, bytes are read from the file offset that is associated with the file descriptor. A successful read increments the file offset by the number of bytes that are read.

For files where no seek operation is possible, there is no file offset associated with the file descriptor. Reading begins at the current position in the file.

- **Number of bytes read:** When a read request completes, the Return\_value field shows the number of bytes that were actually read — a number less than or equal to the number of bytes that were requested. Following are some reasons why the number of bytes that are read might be less than the number of bytes that were requested:
  - Fewer than the requested number of bytes remained in the file; the end of file was reached before all requested bytes were read.
  - The service was interrupted by a signal after some, but not all, of the requested bytes were read. (If no bytes were read, the return value is set to -1 and an error is reported.)
  - The file is a pipe, FIFO, or special file, and fewer bytes than requested were available for reading.

There are several reasons why a read request might complete successfully with no bytes read — that is, with Return\_value set to 0. For example, zero bytes are read in these cases:

- The service specified that zero bytes were to be read.
  - The starting position for the read was at or beyond the end of the file.
  - The file that is being read is a FIFO file or a pipe, and no process has the pipe open for writing.
  - The file that is being read is a subsidiary pseudoterminal, and a zero-length canonical line was written to the manager pseudoterminal.
- **Nonblocking:** If a process has a pipe open for reading with nonblocking specified, a request to read from the file ends with a return value of -1 and a "Resource temporarily unavailable" return code. But if nonblocking is not specified, the read request is blocked (does not return) until some data is written, or until the pipe is closed by all other processes that have the pipe open for writing.

Manager and subsidiary pseudoterminals also operate this way, except that how they act depends on how they were opened. If the manager or the subsidiary is opened blocking, the reads are blocked if there is no data. If they are opened nonblocking, EAGAIN is returned if there is no data.

- **SIGTTIN processing:** The readv service causes signal SIGTTIN to be sent under the following conditions:
  - The process is attempting to read from its controlling terminal, and
  - The process is running in a background process group, and
  - The SIGTTIN signal is not blocked or ignored, and
  - The process group of the process is not orphaned.

If these conditions are met, SIGTTIN is sent. If SIGTTIN has a handler, the handler gets control and the read ends with a return code of EINTR. If SIGTTIN is set to default, the process stops in the read and continues when the process is moved to the foreground.

## Related services

- [“writev \(BPX1WRV, BPX4WRV\) — Write data from a set of buffers ” on page 1036](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1RDV \(readv\) example” on page 1279](#).

# realpath (BPX1RPH, BPX4RPH) – Resolve a path name

## Function

The realpath service derives, from the pathname that is pointed to by Pathname, an absolute pathname that names the same file, whose resolution does not involve dot (.), dot-dot (..), or symbolic links.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1RPH):</b>	31-bit
<b>AMODE (BPX4RPH):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RPH, (Pathname_length,
               Pathname,
               Resolved_name_length,
               Resolved_name,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RPH with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the pathname that is to be resolved.

### Pathname

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the pathname that is to be resolved. The length of this field is specified in Pathname\_length.

Pathnames can begin with or without a slash.

- A path name that begins with a slash is an absolute pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative pathname. The search for the file starts at the working directory.

**Resolved\_name\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the buffer to which the realpath service returns an absolute pathname without dot (.), dot-dot (..), or symbolic links. Resolved\_name\_length must be large enough to accommodate the actual length of an absolute pathname, plus one (for the terminating null). A length of zero has special meaning; see the usage notes.

**Resolved\_name**

Parameter supplied and returned

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Resolved\_name\_length parameter

The name of the buffer that is to hold the absolute pathname that is to be generated for the input Pathname. The length of this field is specified in Resolved\_name\_length.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the realpath service returns the length of the pathname that is in the buffer, if the request is successful; or -1, if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the realpath service stores the return code. The realpath service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The realpath service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search one of the components of Pathname.
<b>EINVAL</b>	There was a parameter error; for example, Resolved_name_length is not valid. The following reason codes can accompany the return code: JRBuffLenInvalid and JRBadAddress.
<b>EIO</b>	An input/output error occurred.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters; or a component of Pathname is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file named Pathname was found; or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of Pathname is not a directory.
<b>ENOMEM</b>	Insufficient storage space is available. The following reason code can accompany the return code: JRNoStorage.
<b>ERANGE</b>	The specified Resolved_name_length is less than the length of the pathname that was generated for the input Pathname. The specified Resolved_name_length is zero, and the length of the path name that was generated for the input Pathname is larger than PATH_MAX bytes. The following reason code can accompany the return code: JrBuffTooSmall.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the realpath service stores the reason code. The realpath service returns a Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. PATH\_MAX plus 1 for the terminating null is a reasonable value for Resolved\_name\_length and for the size of Resolved\_name.
2. If a Resolved\_name\_length value of zero is passed to this service, the generated path name is stored, up to a maximum of PATH\_MAX bytes, in the buffer that is pointed to by Resolved\_name. Resolved\_name is assumed to be of sufficient size to contain the path name that is derived by the realpath service. If the generated path name is larger than PATH\_MAX, the return value is -1 and Return\_code is ERANGE.

**Related services**

- [“getcwd \(BPX1GCW, BPX4GCW\) — Get the path name of the working directory”](#) on page 260
- [“pathconf \(BPX1PCF, BPX4PCF\) — Determine configurable path name variables using a path name”](#) on page 535
- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ”](#) on page 913

## Characteristics and restrictions

None.

## Examples

See “BPX1RPH (realpath) example” on page 1283.

# recv (BPX1RCV, BPX4RCV) – Receive data on a socket and store it in a buffer

---

## Function

The recv callable service receives data on a socket and stores it in a buffer. If no messages are available at the socket, the service either waits for a message to arrive, or fails with EWOULDBLOCK – depending on whether the socket has been defined as blocking or nonblocking, and whether the SO\_RCVTIMEO socket option is in effect.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE: (BPX1RCV)</b>	31-bit
<b>AMODE: (BPX4RCV)</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RCV, (Socket_descriptor,
               Buffer_length,
               Buffer,
               Buffer_alet,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RCV with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

### Type:

Integer

**Length:**

Fullword

The name of a fullword that contains the socket file descriptor for which the receive is to be done.

**Buffer\_length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length of Buffer.

**Buffer**

Supplied parameter

**Type:**

Character

**Length:**

Length specified by Buffer\_length.

The name of a field into which the data is received.

**Buffer\_alet**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the ALET for Buffer. You should specify a Buffer\_alet of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

**Flags**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains information about how the data is to be received. See [“BPXYMSGF — Map the message flags” on page 1107](#) for more information about the format of this field.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recv service returns one of the following:

- The number of bytes received into the buffer, if the request is successful. A value of 0 indicates that the connection is closed.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recv service stores the return code. The recv service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The recv service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EBADF</b>	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINTR</b>	A signal interrupted the recv ( ) function before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	The socket is marked shutdown for read. The following reason code can accompany the return code: JRSocketClosed.
<b>EIO</b>	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EWouldBlock</b>	<ul style="list-style-type: none"> <li>• The socket is marked nonblocking and no data is waiting to be received, or the SO_RCVTIMEO timeout value was reached before data was available.</li> <li>• The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. T             <ul style="list-style-type: none"> <li>– If AT-TLS is being used to provide transparent TLS/SSL support for a TCP socket and EWouldBlock is returned on a blocking socket, reissue the select call or receive call to continue. For more information about AT-TLS and determining whether a TCP connection is using AT-TLS, see <a href="#">Application Transparent Transport Layer Security (AT-TLS)</a> in <i>z/OS Communications Server: IP Programmer's Guide and Reference</i>.</li> </ul> </li> </ul> <p>The following reason codes can accompany the return code: JRTimeout, JRWouldBlock.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recv service stores the reason code. The recv service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



## Usage notes

1. The recv callable service applies only to connected sockets. It can be used with datagram or stream sockets. For datagram sockets, the recv service returns the entire datagram that was sent, providing that the datagram fits into the specified buffers. The excess is discarded. For stream sockets, data is not discarded. Multiple invocations of the recv service may be needed to return all the data.
2. See Chapter 3, “Callable services for SRB mode routines,” on page 1041 for more information about programming considerations for SRB mode.

## Related services

“send (BPX1SND, BPX4SND) — Send data on a socket ” on page 726

## Characteristics and restrictions

None.

## Examples

See “BPX1RCV (recv) example” on page 1278.

# recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer

---

## Function

The recvfrom callable service receives data on a socket and stores it in a buffer. It can be used by an application program to receive data from sockets. When no data is available at the socket, the service either waits for data to arrive, or returns an EWOULDBLOCK — depending on whether the socket is defined as blocking or nonblocking, and whether the SO\_RCVTIMEO socket option is in effect.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RFM)</b>	31-bit task or SRB mode
<b>AMODE (BPX4RFM)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RFM, (Socket_descriptor,
               Buffer_length,
               Buffer,
               Buffer_alet,
```

```

Flags,
Sockaddr_length,
Sockaddr,
Return_value,
Return_code,
Reason_code)

```

AMODE 64 callers use BPX4RFM with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the recvfrom is to be done.

### Buffer\_length

Supplied and returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the length of Buffer.

### Buffer

Supplied parameter

#### Type:

Character

#### Length:

Length specified by Buffer\_length

The name of a field into which the data is to be received.

### Buffer\_alet

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the ALET for Buffer. You should specify a Buffer\_alet of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### Flags

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains information about how the data is to be received. See [“BPXYMSGF — Map the message flags” on page 1107](#) for more information about the format of this field.

**Sockaddr\_length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that, on input, contains the length of the Sockaddr buffer. On return, this field specifies the size required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information contained in Sockaddr is truncated to the length supplied on input. The value in this field should be less than 4096 bytes (4KB) in length, and should represent the maximum possible length of the Sockaddr on output.

**Sockaddr**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length specified by Sockaddr\_length

The name of a buffer area that, on return, contains the socket address of the sender of the data. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the format of this field.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recvfrom service returns one of the following:

- The number of bytes received into the buffer, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recvfrom service stores the return code. The recvfrom service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The recvfrom service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EBADF</b>	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINTR</b>	A signal interrupted the recvfrom function before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	The socket is marked shutdown for read. The following reason codes can accompany the return code: JRSocketCallParmError, JRSocketClosed.

Return code	Explanation
<b>EIO</b>	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EWouldBLOCK</b>	<ul style="list-style-type: none"> <li>• The socket is marked nonblocking and no data is waiting to be read, or the SO_RCVTIMEO timeout value was reached before data was available.</li> <li>• The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. <ul style="list-style-type: none"> <li>– If AT-TLS is being used to provide transparent TLS/SSL support for a TCP socket and EWouldBLOCK is returned on a blocking socket, reissue the select call or receive call to continue. For more information about AT-TLS and determining whether a TCP connection is using AT-TLS, see <a href="#">Application Transparent Transport Layer Security (AT-TLS)</a> in <i>z/OS Communications Server: IP Programmer's Guide and Reference</i>.</li> </ul> </li> </ul> <p>The following reason codes can accompany the return code: JRTimeout, JRWouldBlock.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recvfrom service stores the reason code. The recvfrom service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The recvfrom callable service can be used with datagram or stream sockets. For datagram sockets, it returns the entire datagram that was sent, providing that the datagram fits into the specified buffer. The excess is discarded. For stream sockets, data is not discarded. Multiple invocations of recvfrom may be needed to return all the data.
2. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

**Related services**

[“sendto \(BPX1STO, BPX4STO\) — Send data on a socket ” on page 738](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1RFM \(recvfrom\) example” on page 1280](#).

## recvmsg (BPX2RMS, BPX4RMS) – Receive messages on a socket and store them in message buffers

---

### Function

The recvmsg callable service receives messages on a socket and stores them in a set of buffers. The socket can be either connected or unconnected. If no messages are available at the socket, the service either waits for a message to arrive, or returns an EWOULDBLOCK – depending on whether the socket is defined as blocking or nonblocking, and whether the SO\_RCVTIMEO socket option is in effect.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX2RMS):</b>	31-bit
<b>AMODE (BPX4RMS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX2RMS, (Socket_descriptor,
               Message_hdr,
               Flags,
               Iov_alet,
               Iov_buffer_alet,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RMS with the same parameters. All addresses in the Message\_hdr structure are doublewords.

### Parameters

#### Socket\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the recvmsg is to be done.

#### Message\_hdr

Supplied parameter

**Type:**

Structure

**Length:**

The length of BPXYMSGH

The name of a field that contains the message header, which describes how the message is to be received. See [“BPXYMSGH — Map the message header”](#) on page 1109 for more information about the format of this field. In 64-bit mode, Message\_hdr contains doubleword pointer subfields, and points to an Iov\_struct structure that contains doubleword pointer and length subfields.

**Flags**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains information about how the data is to be received. See [“BPXYMSGF — Map the message flags”](#) on page 1107 for more information about the format of this field.

**Iov\_alet**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the ALET for the IOV structure that is specified in Message\_hdr.

**Iov\_buffer\_alet**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the ALET for the buffers that are pointed to by the IOV structure that is specified in Message\_hdr.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recvmsg service returns one of the following:

- The number of bytes that were read into the buffers, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the recvmsg service stores the return code. The recvmsg service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The recvmsg service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EBADF</b>	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINTR</b>	A signal interrupted the recvmsg service before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	The socket is marked shutdown for read; or incorrect data was received as a parameter. The following reason codes can accompany the return code: JRInvalidMsg, JRSocketClosed, JRSocketCallParmError.
<b>EIO</b>	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EWouldBlock</b>	<ul style="list-style-type: none"> <li>• The socket is marked nonblocking and no data is waiting to be read, or the SO_RCVTIMEO timeout value was reached before data was available.</li> <li>• The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. <ul style="list-style-type: none"> <li>– If AT-TLS is being used to provide transparent TLS/SSL support for a TCP socket and EWouldBlock is returned on a blocking socket, reissue the select call or receive call to continue. For more information about AT-TLS and determining whether a TCP connection is using AT-TLS, see <a href="#">Application Transparent Transport Layer Security (AT-TLS)</a> in <i>z/OS Communications Server: IP Programmer's Guide and Reference</i>.</li> </ul> </li> </ul> <p>The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.</p>

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the recvmsg service stores the reason code. The recvmsg service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

#### Usage notes

1. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.
2. The BPX2RMS/BPX4RMS call supersedes the BPX1RMS call, which is still available for migration purposes only.
3. If the sendmsg security label is not equivalent to the recvmsg security label when access rights are passed on the sendmsg, the new descriptors are not created.

## rename (BPX1REN, BPX4REN)

4. The number of buffers that are pointed to by the IOV structure in Message\_hdr may not exceed IOV\_MAX (defined in “BPXYIOV — Map the I/O vector structure” on page 1098).
5. For the IP\_PKTINFO ancillary data item, the program has to call setsockopt() or BPXIOPT(Sock#OptOptSetSockOpt) with the IP\_RECVPKTINFO option to have the TCPIP stack pass the client's return information in the IN\_PKTINFO structure as an ancillary data item on the recvmsg() or BPX2RMS call. That IP\_PKTINFO data is used unchanged on the subsequent sendmsg() or BPX2SMS calls to have the reply flow out the same interface through which the request arrived.

### Related services

“sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket ” on page 734

### Characteristics and restrictions

None.

### Examples

See “BPX2RMS (recvmsg) example” on page 1282.

## rename (BPX1REN, BPX4REN) — Rename a file or directory

---

### Function

The rename callable service changes the name of a file or a directory.

### Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1REN)	31-bit
AMODE (BPX4REN)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1REN,(Old_name_length,  
              Old_name,  
              New_name_length,  
              New_name,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4REN with the same parameters.



**Parameters****Old\_name\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of the path name of the file or directory that is to be renamed.

**Old\_name**

Supplied parameter.

**Type**

Character string

**Character set.**

No restriction

**Length**

Specified by the Old\_name\_length parameter

The name of a field of length Old\_name\_length that contains the name of the existing file or directory.

**New\_name\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of the path name that is to be given to the existing file or directory.

**New\_name**

Supplied parameter.

**Type**

Character string

**Character set.**

No restriction

**Length**

Specified by the New\_name\_length parameter

The name of a field, of length New\_name\_length that contains the new path name of the file or directory.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the rename service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the rename service stores the return code. The rename service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The rename service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• The process did not have search permission on some component of the old or new path name or did not have write permission on the parent directory of the file or directory that is to be renamed.</li> <li>• The S_ISVTX flag is set for the directory that contains Old_name. The caller is not the owner of Old_name or the owner of the parent directory. The caller also does not have appropriate privileges (see <a href="#">“Authorization” on page 8</a>).</li> <li>• New_name refers to an existing file. The S_ISVTX flag is set for the directory that contains New_name and the caller is not the owner of New_name or the owner of the parent directory. The caller also does not have appropriate privileges.</li> </ul>
<b>EAGAIN</b>	<p>One of the files or directories was temporarily unavailable. The following reason code can accompany the return code: JRInvalidVnode.</p>
<b>EBUSY</b>	<p>Old_name and New_name specify directories but one of them cannot be renamed because it is in use as a root or a mount point, or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation. The following reason code can accompany the return code: JRIisFSRoot.</p>
<b>EINVAL</b>	<p>This error is returned for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• Old_name is part of the path name prefix of New_name</li> <li>• Old_name refers to either . (dot) or . . (dot-dot).</li> <li>• New_name refers to either . (dot) or . . (dot-dot).</li> </ul> <p>The following reason codes can accompany the return code: JRDotOrDotDot and JROldPartOfNew.</p>
<b>EISDIR</b>	<p>New_name identifies a directory, but Old_name is not a directory. The following reason code can accompany the return code: JRNewIsDir.</p>
<b>ELOOP</b>	<p>A loop exists in symbolic links that were encountered during resolution of the Old_name or New_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Old_name or New_name.</p>
<b>ENAMETOOLONG</b>	<p>Old_name or New_name is longer than 1023 bytes, or a component of one of those names is longer than 255 bytes. Name truncation is not supported.</p>
<b>ENOENT</b>	<p>No file or directory name Old_name was found, or either Old_name or New_name was not specified. The following reason code can accompany the return code: JROldNoExist.</p>
<b>ENOSPC</b>	<p>The directory that is intended to contain New_name cannot be extended.</p>

Return_code	Explanation
<b>ENOTDIR</b>	A component of either path name prefix is not a directory or Old_name is a directory and New_name is a file that is not a directory. The following reason code can accompany the return code: JRNewNotDir.
<b>ENOTEMPTY</b>	New_name specifies a directory, but the directory is not empty. It contains files or subdirectories.
<b>EROFS</b>	Performing the requested service would make it necessary to write on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.
<b>EXDEV</b>	Old_name and New_name identify files or directories on different file systems. Renaming across file systems is not supported. The following reason code can accompany the return code: JRDiffFileSets.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the rename service stores the reason code. The rename service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The rename service changes the name of a file or directory from Old\_name to New\_name. When the renaming request completes successfully, the change and modification times for the parent directories of Old\_name and New\_name are updated.

For renaming to succeed, the calling process needs write permission for the directory that contains Old\_name and the directory that contains New\_name. If Old\_name and New\_name are the names of directories, the caller does not need write permission for the directories themselves.

2. If the S\_ISVTX flag is set for the directory that contains Old\_name, one of the following conditions must be true, or the request will fail with EACCES:
  - The caller is the owner of the file named Old\_name.
  - The caller is the owner of the parent directory that contains Old\_name.
  - The caller has appropriate privileges (see [“Authorization”](#) on page 8).

If the S\_ISVTX flag is set for the directory that contains New\_name, where New\_name refers to an existing file, one of the following conditions must be true, or the request will fail with EACCES:

- The caller is the owner of the file named New\_name.
- The caller is the owner of the parent directory that contains New\_name.
- The caller has appropriate privileges.

3. Renaming files:
  - If Old\_name and New\_name are links that refer to the same file, the rename service simply returns successfully.
  - If Old\_name is the name of a file, New\_name must also name a file, not a directory. If New\_name is an existing file, it is unlinked, and then the file that is specified as Old\_name is given New\_name. The path name New\_name always stays in existence. At the beginning of the operation, New\_name refers to its original file, and at the end, it refers to the file that used to be Old\_name.

- The rename will fail with EBUSY if New\_name refers to an existing file that is open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#) for details about the NFS share reservations.

#### 4. Renaming directories:

- If Old\_name is the name of a directory, New\_name must also name a directory, not a file. If New\_name is an existing directory, it must be empty, containing no files or subdirectories. If it is empty, it is removed, as described in [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory” on page 703](#).
- New\_name cannot be a directory under Old\_name; that is, the old directory cannot be part of the path name prefix of the new one.

### Related services

- [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file” on page 383](#)
- [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor” on page 387](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory” on page 703](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

### Characteristics and restrictions

None.

### Examples

See [“BPX1REN \(rename\) example” on page 1280](#) and [“BPX4REN \(rename\) example” on page 1373](#).

## renameat (BPX1RNA, BPX4RNA) — Rename a file using the at directory relative to a directory file descriptor

---

### Function

The renameat2 service is identical to the rename service except that it uses directory file descriptors as the starting location for the path searches for the old and new path names, with optional flags that indicate how the path should be resolved.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RNA)</b>	31-bit
<b>AMODE (BPX4RNA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation**

**Environment**

**Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1RNA, (Old_name_length,  
               Old_name,  
               New_name_length,  
               New_name,  
               OldDirFD,  
               NewDirFD,  
               Flags,  
               Return value,  
               Return code,  
               Reason code)
```

AMODE 64 callers use BPX4RNA with the same parameters.

**Parameters**

**Oldname\_length**

Supplied parameter.

**Type**

Integer. The allowed range of lengths is the same as path name lengths on other file system calls.

**Old\_name**

Supplied parameter.

**Type**

Character string.

**Length**

Variable.

Pointer to a string that has the path name of the file to link to.

**New\_name\_length**

Supplied parameters.

**Type**

Integer.

**Integer**

Fullword.

The name of a fullword that contains the length of the path name that is to be given to the existing file or directory.

**New\_name**

Supplied parameter.

**Type**

Character string.

**Length**

Variable.

Specified by the new\_name\_length parameter.

**OldDirFD**

Supplied parameter.

**Type**

File descriptor.

**Length**

Fullword.

The name of a fullword that contains a file descriptor for the starting directory to search for the existing file or the special value AT\_FDCWD, which means to use the current working directory as the starting point. If the Old\_name is absolute, this parameter is ignored.

**NewDirFD**

Supplied parameter.

**Type**

File descriptor.

**Length**

Fullword.

The name of a fullword that contains a file descriptor for the directory to contain the newly created link, or the special value AT\_FDCWD, which means to use the current working directory as the starting point. If the New\_name is absolute, this parameter is ignored.

**Flags**

Supplied parameters.

**Type**

Integer.

Can contain zero or the following value.

**RENAME\_NOREPLACE**

Does not overwrite the New\_name of the rename. Returns an error if New\_name already exists.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the renameat service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the renameat service stores the return code. The renameat service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The renameat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCESS</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• The process did not have search permission on some component of the old or new path name or did not have write permission on the parent directory of the file or directory that is to be renamed.</li> <li>• The S_ISVTX flag is set for the directory that contains Old_name. The caller is not the owner of Old_name or the owner of the parent directory. The caller also does not have appropriate privileges (see <a href="#">“Authorization” on page 8</a>).</li> <li>• New_name refers to an existing file. The S_ISVTX flag is set for the directory that contains New_name and the caller is not the owner of New_name or the owner of the parent directory. The caller also does not have appropriate privileges.</li> </ul>
<b>EAGAIN</b>	One of the files or directories was temporarily unavailable. The following reason code can accompany the return code: JRInvalidVnode.
<b>EBADF</b>	Old_name (New_name) is relative but olddirfd (newdirfd) is not a valid file descriptor.
<b>EBUSY</b>	Old_name and New_name specify directories but one of them cannot be renamed because it is in use as a root or a mount point, or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation. The following reason code can accompany the return code: JRIisFSRoot.
<b>EEXIST</b>	flags contains RENAME_NOREPLACE and New_name already exists.
<b>EINVAL</b>	<p>This error is returned for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• Old_name is part of the path name prefix of New_name.</li> <li>• Old_name refers to either . (dot) or . . (dot-dot).</li> <li>• New_name refers to either . (dot) or . . (dot-dot).</li> <li>• An invalid flag was specified in Flags.</li> </ul>
<b>EISDIR</b>	New_name identifies a directory, but Old_name is not a directory. The following reason code can accompany the return code: JRNewIsDir.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Old_name or New_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Old_name or New_name.
<b>ENAMETOOLONG</b>	Old_name or New_name is longer than 1023 bytes, or a component of one of those names is longer than 255 bytes. Name truncation is not supported.
<b>ENOENT</b>	No file or directory name Old_name was found, or either Old_name or New_name was not specified. The following reason code can accompany the return code: JROldNotExist.
<b>ENOSPC</b>	The directory that is intended to contain New_name cannot be extended.
<b>ENOTDIR</b>	<p>A component of either path name prefix is not a directory or Old_name is a directory and New_name is a file that is not a directory. The following reason code can accompany the return code: JRNewNotDir.</p> <p>Old_name is relative and OldDirFD is a file descriptor referring to a file other than a directory, or similar for New_name and NewDirFD.</p>
<b>ENOTEMPTY</b>	New_name specifies a directory, but the directory is not empty. It contains files or subdirectories.

Return_code	Explanation
<b>EROFS</b>	Performing the requested service would make it necessary to write on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.
<b>EXDEV</b>	Old_name and New_name identify files or directories on different file systems. Renaming across file systems is not supported. The following reason code can accompany the return code: JRDiffFileSets.

### Reason\_code

Returned parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword in which the renameat service stores the reason code. The link service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The renameat service changes the name of a file or directory from Old\_name to New\_name. When the renaming request completes successfully, the change and modification times for the parent directories of Old\_name and New\_name are updated.

For renaming to succeed, the calling process needs write permission for the directory that contains Old\_name and the directory that contains New\_name. If Old\_name and New\_name are the names of directories, the caller does not need write permission for the directories themselves.

2. If the S\_ISVTX flag is set for the directory that contains Old\_name, one of the following conditions must be true, or the request will fail with EACCES:
  - The caller is the owner of the file named Old\_name.
  - The caller is the owner of the parent directory that contains Old\_name.
  - The caller has appropriate privileges (see “Authorization” on page 8).

If the S\_ISVTX flag is set for the directory that contains New\_name, where New\_name refers to an existing file, one of the following conditions must be true, or the request will fail with EACCES:

- The caller is the owner of the file named New\_name.
- The caller is the owner of the parent directory that contains New\_name.
- The caller has appropriate privileges.

3. Renaming files:
  - If Old\_name and New\_name are links that refer to the same file, the renameat service simply returns successfully.
  - If Old\_name is the name of a file, New\_name must also name a file, not a directory. If New\_name is an existing file, it is unlinked, and then the file that is specified as Old\_name is given New\_name. The path name New\_name always stays in existence. At the beginning of the operation, New\_name refers to its original file, and at the end, it refers to the file that used to be Old\_name.
  - The renameat service will fail with EBUSY if New\_name refers to an existing file that is open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to “open (BPX1OPN, BPX4OPN) — Open a file” on page 511 for details about the NFS share reservations.

4. Renaming directories:



- If Old\_name is the name of a directory, New\_name must also name a directory, not a file. If New\_name is an existing directory, it must be empty, containing no files or subdirectories. If it is empty, it is removed, as described in [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory”](#) on page 703.
- New\_name cannot be a directory under Old\_name; that is, the old directory cannot be part of the path name prefix of the new one.

## Related services

- [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory”](#) on page 690
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry”](#) on page 966
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor”](#) on page 969

## Characteristics and restrictions

None.

## Examples

See [“BPX1RNA \(renameat\) example”](#) on page 1282 and [“BPX4RNA \(renameat\) example”](#) on page 1375.

# resource (BPX1RMG, BPX4RMG) — Measure resources

---

## Function

The resource callable service gets system-wide resource measurement data from the kernel address space.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RMG)</b>	31-bit
<b>AMODE (BPX41RMG)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RMG, (Data_area_length,
               Data_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4RMG with the same parameters.

## Parameters

### Data\_area\_length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of Data\_area, which the resource service is to fill with resource measurement information.

### Data\_area

Supplied parameter

**Type:**

Structure

**Length:**

Specified by the Data\_area\_length parameter

The name of a field of length Data\_area\_length, which the resource service is to fill with resource measurement information. This field is mapped by the macro BPXYRMON. For the structure of Data\_area, see [“BPXYRMON — Map resource monitor data” on page 1142](#).

### Return\_value

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the resource service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the resource service stores the return code. The resource service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The resource service can return the following value in the Return\_code parameter:

Return_code	Explanation
EINVAL	Incorrect argument.

### Reason\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the resource service stores the reason code. The resource service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. Some values that are returned by the resource service are continually wrapping counters. At the first call to the resource service, these values should be stored. At subsequent calls, the growth in these values should be calculated by the caller. The following list describes the normal use of wrapping counters that are returned by the resource service:
  - a. A first call to the resource service returns the current value. (For example, X'FFFFFFD0' is returned for a system call count.)
  - b. After some time interval expires, a second call to the resource service returns the new value. (For example, X'00000028' is returned for a system call count.)
  - c. At this point, the increase in the counter can be calculated by the calling application. (In this case, we can calculate that X'58', or 88, system calls have been processed between the first resource service request and the second.)

## Characteristics and restrictions

None.

## Examples

See [“BPX1RMG \(resource\) example” on page 1281](#).

# rewinddir (BPX1RWD, BPX4RWD) – Reposition a directory stream to the beginning

---

## Function

The rewinddir callable service "rewinds," or resets, to the beginning of, an open directory. The next call to the readdir service reads the first entry in the directory.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1RWD):</b>	31-bit
<b>AMODE (BPX4RWD):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RWD (Directory_file_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RWD with the same parameters.

## Parameters

### Directory\_file\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened (see [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory”](#) on page 523).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the rewinddir service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the rewinddir service stores the return code. The rewinddir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The rewinddir service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The Directory_file_descriptor parameter does not represent an open directory. The following reason code can accompany the return code: JRRwdFileNotDir.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the rewinddir service stores the reason code. The rewinddir service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

If the contents of the directory that you specify have changed since the directory was opened, a call to the rewinddir service resets the pointer into the directory to the beginning. A subsequent call to the readdir service reads from the start of the directory and obtains the new contents.

## Related services

- [“closedir \(BPX1CLD, BPX4CLD\) — Close a directory” on page 124](#)
- [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory” on page 523](#)
- [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory ” on page 656](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1RWD \(rewinddir\) example” on page 1283](#).

# rmdir (BPX1RMD, BPX4RMD) — Remove a directory

---

## Function

The rmdir callable service removes a directory. The directory must be empty.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1RMD)</b>	31-bit
<b>AMODE (BPX4RMD)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1RMD,(Directory_name_length,
              Directory_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RMD with the same parameters.

**Parameters**

**Directory\_name\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of Directory\_name.

**Directory\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Directory\_name\_length parameter

The name of a field that contains the pathname of the directory to be removed. The length of this field is specified in Directory\_name\_length.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the rmdir service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the rmdir service stores the return code. The rmdir service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The rmdir service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EACCES	One of the following conditions occurred: <ul style="list-style-type: none"><li>• The process did not have search permission for some component of Directory_name, or did not have write permission for the directory that contains the directory that is to be removed.</li><li>• The S_ISVTX flag is set for the parent directory of the directory that is to be removed, and the caller is not the owner of that directory or the owner of the parent directory, nor does the caller have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).</li></ul>

Return_code	Explanation
<b>EBUSY</b>	The directory cannot be removed, because it is being used by a process. The following reason code can accompany the return code: JRRootNode.
<b>EINVAL</b>	The argument that was supplied was incorrect. Examples of incorrect arguments are dot and dot-dot. The following reason code can accompany the return code: JRDotOrDotDot.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Directory_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Directory_name.
<b>ENAMETOOLONG</b>	The name of the directory is longer than 1023 characters; or some component of the pathname is longer than 255 characters. This could be as a result of encountering a symbolic link during resolution of Directory_name, where the substituted string is longer than 1023 characters.
<b>ENOENT</b>	The directory that was specified by Directory_name was not found; or no Directory_name parameter was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of Directory_name is not a directory. The following reason code can accompany the return code: JRPathNotDir.
<b>ENOTEMPTY</b>	The directory contains files or subdirectories.
<b>EROFS</b>	The directory that is to be removed is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the rmdir service stores the reason code. The rmdir service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The directory must be empty.
2. If the directory is successfully removed, the change and modification times for the parent directory are updated.
3. If the link count of the directory becomes zero and no process has the directory open, the directory itself is deleted. The space that is occupied by the directory is freed for new use, and the contents of the file are lost.
4. If any process has the directory open when the last link is removed, the directory itself is not removed until the last process closes the directory. New files cannot be created under a directory after the last link is removed, even if the directory is still open.
5. If the S\_ISVTX flag is set for the parent directory of the directory that is to be removed, one of the following conditions must be true, or the request will fail with EACCES:
  - The caller is the owner of the directory to be removed
  - The caller is the owner of the parent directory
  - The caller has appropriate privileges (see [“Authorization”](#) on page 8)

**Related services**

- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1RMD \(rmdir\) example” on page 1281.](#)

## **select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues**

---

**Function**

The select/selectex callable service checks the I/O status of multiple open file descriptors and message queues. The file descriptors can be for character special files, pipes, sockets, or files.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SEL)</b>	31-bit
<b>AMODE (BPX4SEL)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SEL, (Number_msgsfds,
               Read_list_length,
               Read_list,
               Write_list_length,
               Write_list,
               Exception_list_length,
               Exception_list,
               Timeout_pointer,
               Ecb_pointer,
               User_option_field,
               Return_value,
```



```
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4SEL with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Number\_msgsfds

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword of which the first halfword (the high-order 16 bits) contains the number of message queues and the second halfword (the low-order 16 bits) contains the number of file descriptors.

The number of file descriptors should be the highest file descriptor that is being checked for status, plus 1.

For example, if you are interested in the I/O status of file descriptors 5 and 9, the second halfword of Number\_msgsfds would be 10. Ten is the number of file descriptors that are contained in each of the bit sets (fd 0 through 9 equals 10 fds), and 10 is the highest file descriptor that is being checked, plus 1 (9 plus 1 equals 10). If you want to check file descriptors for status along with message queues, the highest file descriptor you can specify is 2047.

The number of message queues indicates the number of elements (queue IDs) in each of the arrays contained in Read\_list, Write\_list, and Exception\_list. For example, if you specify a value of 10 in the first halfword of Number-msgsfds, it is expected that arrays of 10 elements each are given in Read\_list, Write\_list, and Exception\_list. If you specify a value of 0, it is assumed that no arrays are given and that no message queues are to be checked. The maximum number of message queues that you can specify is 32 767.

**Note:** In order to select on descriptor numbers higher than 65 534, the descriptor limit of the process must be at least 65 536, and one of the bit lists that is passed must be at least 8192 bytes long. When both of these facts are true, and the fullword value is between 65 536 and the system descriptor maximum, the entire fullword parameter will be assumed to represent the number of file descriptors, and no message queues will be processed.

### Read\_list\_length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length, in bytes, of the Read\_list. The length is actually the sum of the length (rounded up to a multiple of 4 bytes) of the bit set specifying file descriptors and the length of the array of message queue identifiers. When both file descriptors and message queues are specified, this field should contain a value greater than 256 bytes. If 0 is specified, the Read\_list is not checked by the select service.

### Read\_list

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length specified by Read\_list\_length

The name of a structure that contains the bit set for the specified file descriptors and/or the array of message queue identifiers. The bit set must be padded with extra bytes, if necessary, to round up its length to the next multiple of 4 bytes. The bits in the bit set should be turned on for the corresponding descriptors to be checked for reading. The format of the bits can be specified with the `User_option` field. On return, the bits that are set indicate the descriptors that are ready for reading.

If `Read_list` contains both a bit set and an array of message queue identifiers, the bit set must be 256 bytes in length. If only file descriptors are to be checked, the bit set can have any valid size.

Each element of the array of message queue identifiers is 4 bytes in length. Elements with a value of -1 are acceptable and are ignored. On return, the array is altered such that message queue identifiers that do not meet the criterion are replaced with a value of -1.

**Write\_list\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length, in bytes, of the `Write_list`. The length is actually the sum of the length (rounded up to a multiple of 4 bytes) of the bit set specifying file descriptors and the length of the array of message queue identifiers. When both file descriptors and message queues are specified, this field should contain a value greater than 256 bytes. If 0 is specified, the `Write_list` is not checked by the select service.

**Write\_list**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length specified by `Write_list_length`

The name of a structure that contains the bit set for the specified file descriptors and/or the array of message queue identifiers. The bit set must be padded with extra bytes, if necessary, to round up its length to the next multiple of 4 bytes. The bits in the bit set should be turned on for the corresponding descriptors to be checked for writing. The format of the bits can be specified with the `User_option` field. On return, the bits that are set indicate the descriptors that are ready for writing.

If `Write_list` contains both a bit set and an array of message queue identifiers, the bit set must be 256 bytes in length. If only file descriptors are to be checked, the bit set can have any valid size.

Each element of the array of message queue identifiers is 4 bytes in length. Elements with a value of -1 are acceptable and are ignored. On return, the array is altered such that message queue identifiers that do not meet the criterion are replaced with a value of -1.

**Exception\_list\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length in bytes of the `Exception_list`. The length is actually the sum of the length (rounded up to a multiple of 4 bytes) of the bit set specifying file descriptors and the length of the array of message queue identifiers. When both file descriptors and message queues are specified, this field should contain a value greater than 256 bytes. If 0 is specified, the `Exception_list` is not checked by select.

**Exception\_list**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length specified by Exception\_list\_length

The name of a structure that contains the bit set for the specified file descriptors and/or the array of message queue identifiers. The bit set must be padded with extra bytes, if necessary, to round up its length to the next multiple of 4 bytes. The bits in the bit set should be turned on for the corresponding descriptors to be checked for exceptions. The format of the bits can be specified with the User\_option field. On return, the bits that are set indicate the descriptors that have had exceptions.

If Exception\_list contains both a bit set and an array of message queue identifiers, the bit set must be 256 bytes in length. If only file descriptors are to be checked, the bit set can have any valid size.

Each element of the array of message queue identifiers is 4 bytes in length. Elements with a value of -1 are acceptable and will be ignored. On return, the array is altered such that message queue identifiers that do not meet the criterion are replaced with a value of -1.

**Timeout\_pointer**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains a pointer to a timeout value that controls how the file descriptors are checked:

## 1. Wait indefinitely:

If the timeout\_pointer is zero, the select system call waits (indefinitely) until one of the selected descriptors is ready.

## 2. Wait for a specified period of time:

If the timeout\_pointer is nonzero, it points to a timeout value mapped by the BPXYSELT macro, which contains the number of microseconds and/or seconds to wait for one of the conditions to occur before returning to the caller. The maximum time that can be specified is 31 days. See [“BPXYSELT — Map the timeout value for the select syscall”](#) on page 1145 for more information.

- Microseconds can be a value in the range from 0 to 1 000 000. (1 000 000 microseconds equal 1 second).
- Seconds can be a value in the range from 0 to 2 678 400. (2 678 400 seconds equal 31 days).

**Note:** Microseconds and seconds are added together to determine the timeout value. If the timeout value is more than 0 and less than 300 microseconds, the value is rounded up to 300 microseconds.

## 3. No waiting:

If the timeout value is 0, select returns immediately after checking the selected descriptors; no waiting is done.

**Ecb\_pointer**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

This can be any of the following values:

1. The name of a fullword (doubleword) field that contains a pointer to a user event control block. To specify this usage of Ecb\_pointer, set the high-order bit in Ecb\_pointer to B'0'.

**select/selectex (BPX1SEL, BPX4SEL)**

If a doubleword is used for Ecb\_pointer, the high half must be set to zero (ECBs must be below the bar). In this case, the high-order bit that indicates Ecb\_pointer usage is the high bit in the lower half of the doubleword.

- 2. The name of a fullword (doubleword) field that contains a pointer to a list of ECBs. To specify this usage of Ecb\_pointer, set the high-order bit in Ecb\_pointer B'1'.

The list can contain the pointers for up to 1013 ECBs. The high-order bit of the last pointer in the list must be set to B'1'. If the input Ecb\_pointer is a doubleword, the high half must be zero, and the bit that is checked is the high-order bit of the lower half of the doubleword. If the high-order bit is a 1, the lower half of the doubleword points to a list of Ecb\_pointers. All Ecb\_pointers in the list must be 31-bit pointers.

- 3. The name of a fullword (doubleword) field that contains 0. This indicates that no ECBs are specified.

**User\_option\_field**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

A dual-purpose field that is used as input to specify the format of the read, write, and exception bit lists, and as output to contain the first selected file descriptor that was not supported by the select service.

On input, specify one of the following (the values are defined in [“BPXYSEL — Map the select options”](#) on page 1145):

- SEL#BITSBACKWARD – Bit-backward order by word:

Bits are read from right to left within each word, with the low-order bit on the right and the high-order bit on the left. For example:

Word 1	Word 2	Word 3
31 30 29...3 2 1 0	63 62 61...35 34 33 32	95 94 93...67 66 65 64

In this example, file descriptor 0 is represented by the last bit on the right in Word 1.

- SEL#BITSFORWARD – Bit-forward order by word:

Bits are read from left to right within each word, with the low-order bit on the left and the high-order bit on the right. For example:

Word 1	Word 2	Word 3
0 1 2 3...29 30 31	32 33 34 35...61 62 63	64 65 66.67...93 94 95

In this example, file descriptor 0 is represented by the first bit on the left in Word 1.

On output, the select service returns one of the following:

- -1, if all the selected file descriptors supported the select callable service.
- The first selected file descriptor that did not support the select callable service.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the select service returns one of the following:

- The number of read, write, and exceptional conditions that were found among the given message queues; and the number of read, write, and exceptional conditions that were found among the specified file descriptors. These two values are returned, respectively, in the first halfword and the second halfword of Return\_value. Should the return value for message queues exceed 32 767, only 32 767 is reported. This is to ensure that Return\_value does not appear to be negative. Should the return value for file descriptors be greater than 65 535, only 65 535 is reported.
- 0, if the timeout value expired before any of the conditions were met.
- -1, if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the select service stores the return code. The select service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The select service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINTR</b>	The select service request was interrupted by a signal for the caller.
<b>EINVAL</b>	One of the parameters contains a value that is not correct. The following reason codes can accompany this return code: JRNoLists, JRListTooShort, JRMsOutOfRange, JRInvUserOp, JRSecOutOfRange, JRNoFds, JrNoFdsTooManyQIds, JRTooManyFds, JRListLenBad.
<b>EIO</b>	One of the descriptors in the select mask has become inoperative, and it is being included repeatedly in a select, even though other operations against this descriptor have been failing with EIO. A socket descriptor can become inoperative, for example, if TCP/IP is shut down. When a descriptor fails, a failure from select does not tell you which descriptor has failed. The select call usually succeeds, and the descriptors are reported to you as being ready for whatever events were specified on the select call. When the descriptor is subsequently used on a receive or other operation, you will receive the EIO failure and can then react to the problem with that individual descriptor. In general, you would close() the descriptor and remove it from the next select mask. If the failing return code of the individual descriptor is ignored and an inoperative descriptor is repeatedly selected on and used (even though each time it is used the call fails with EIO), the select call itself will eventually fail with EIO.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the select service stores the reason code. The select service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The key 8 representation in the PSW-key mask (PKM) should not be set off prior to the BPX1SEL call.
2. The bit set for the read\_list, write\_list, and exception\_list is a string of bits such that if X is an element of the set, the bit that represents X is set to 1. For example, if descriptor 1 is to be checked, bit 1 should be turned on in the bit set. Here is how that byte would look:
  - Bit-forward order: B'01000000'.
  - Bit-backward order: B'00000010'.

3. When a positive value is specified for the number of file descriptors:

- At least one bit set (read, write, or exception) must be specified, and its length must be large enough (rounded up to the next multiple of 4) to contain the bit that represents the largest descriptor you specified.
- If more than one bit set is specified, each bit set must be the same length.

For example, if you want to check the read status for file descriptor 59 and the write status for file descriptor 6:

Number of fds = 60

Read\_list\_length = 8

Read\_list = the bit representing fd 59 is set on (see User\_option\_field to determine which bit that would be)

Write\_list\_length = 8

Write\_list = the bit representing fd 6 is set on (see User\_option\_field to determine which bit that would be)

Exception\_list\_length = 0

4. When both the first and second halfwords of Number\_msgsfds contain a positive value, the Read\_list, Write\_list, and Exception\_list must each contain both a bit set and an array of message queue identifiers, unless a value of 0 is specified for its length.

When the fullword value is between 65 536 and the system descriptor maximum, one of the lists that is passed is at least 8 192 bytes long, and the descriptor limit of the process is at least 65 536, the fullword value is considered as the number of descriptors in the lists, and no message queues will be processed.

Suppose you want to check the read status for file descriptors 3 and 5 and the write status for message queues whose identifiers are 7 and 8.

Number of fds = 6 (the largest fd plus 1)

Number of message queues = 2

Read\_list\_length = 264 (256 byte bit set length + 8 byte array length) Read\_list = the 256-byte bit set with appropriate bits set on for fds 3 and 5, followed by a two-element array that contains the value of -1 in both elements.

Write\_list\_length = 264 (same length as for read) Write\_list = the 256-byte bit set with all its bits set off followed by the two-element array that contains the numbers 7 and 8.

Exception\_list\_length = 0

5. You can use the select service as a timer-only function by specifying zero for the Read\_list\_length, Write\_list\_length, and Exception\_list\_length, and by specifying timeout\_pointer and timeout\_value. If you specify zero for timeout\_pointer, the select service blocks forever. If you specify zero for timeout\_value, no blocking is done, and the select service returns immediately to the caller.
6. You can also specify an Ecb\_pointer with the timer only function.
7. Regular files are always ready for reading and writing.

8. When the storage key of the first (or only) ECB matches the caller's PSW key, the kernel performs the wait in the caller's PSW key; otherwise, the kernel performs the wait in the TCB key (TCBPFK). However, if the caller is running in key 0, then the kernel performs the wait in key 0, regardless of the storage key.

## Characteristics and restrictions

None.

## Examples

For an example that uses this callable service, see [“BPX1SEL \(select\) example”](#) on page 1285.

# semctl (BPX1SCT, BPX4SCT) – Perform semaphore control operations

---

## Function

The semctl service provides semaphore control operations. These functions include reading and changing the values of semaphores and removing a set of semaphores from the system.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SCT):</b>	31-bit
<b>AMODE (BPX4SCT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SCT, (Semaphore_ID,
               Semaphore_Number
               Command,
               SValue | Argument_address ( Buffer | Array),
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SCT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Semaphore\_ID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

Specifies the semaphore identifier.

**Semaphore\_Number**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

Specifies the semaphore number. Semaphore\_Number ranges from 0 to Number\_of\_Semaphores - 1. Use with Sem\_GETVAL, Sem\_SETVAL, Sem\_GETNCNT and Sem\_GETZCNT. This argument is ignored for all other commands.

**Command**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword field that indicates the semaphore command that is to be executed. For the structure that contains these constants, see [“BPXYSEM — Map interprocess communication semaphores”](#) on page 1146 and [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098. The values for Command are:

**Sem\_GETVAL**

Returns the value of semval for the requested Semaphore\_Number, if the current process has read permission.

**Sem\_SETVAL**

Sets the semval for the requested Semaphore\_Number to the contents of SValue, if the current process has alter permission. When this Command is successfully executed, the semadj values that correspond to this semaphore for all processes are cleared.

**Sem\_GETPID**

Returns the ID of the most recent process to update the semaphore, if the current process has read permission.

**Sem\_GETNCNT**

Returns the number of threads waiting on the semaphore to become greater than the current value, if the current process has read permission. See [“semop \(BPX1SOP, BPX4SOP\) — Perform semaphore serialization operations”](#) on page 723.

**Sem\_GETZCNT**

Returns the number of threads waiting on the semaphore to become zero, if the current process has read permission. See [“semop \(BPX1SOP, BPX4SOP\) — Perform semaphore serialization operations”](#) on page 723.

**Sem\_GETALL**

Stores all semaphore semvals into the array of halfwords that is pointed to by the Argument\_address parameter, if the current process has read permission. It is the caller's responsibility to ensure that the storage that is allocated for the array is large enough to hold all semaphore elements. The number of semaphore values stored is SEM\_NSEMS, which can be obtained using the Ipc\_STAT command.

**Sem\_SETALL**

Sets semvals according to the array that is pointed to by the Argument\_address parameter, if the current process has alter permission. Each value must be zero or positive. When this Command



is successfully executed, the semadj values that correspond to each specified semaphore in all processes are cleared. It is the caller's responsibility to ensure that the storage that is allocated for the array is large enough for all semaphore elements. The number of semaphore values read is SEM\_NSEMS, which can be obtained using the Ipc\_STAT command.

If IPC\_BINSEM is specified on the semget call, this option should not be used when there is a possibility that other threads could be performing semaphore operations on this semaphore, as there might be no serialization while the semaphore values are being updated.

### **Ipc\_STAT**

Obtains status information about the semaphore that is identified by the Semaphore\_ID parameter, if the current process has read permission. This information is stored in the buffer that is pointed to by the Argument\_address parameter.

### **Ipc\_STAT\_EX**

When the AMODE of the caller is 31, it provides the same output as IPC\_STAT when AMODE is 64 (that is, extended POSIX times are provided). When the AMODE of the caller is 64, IPC\_STAT\_EX and IPC\_STAT are equivalent.

### **Ipc\_SET**

Sets the value of the IPC\_UID, IPC\_GID and IPC\_MODE from the SEMID\_DS data structure that is associated with Semaphore\_ID into the SEMID\_DS structure that is pointed to by Argument\_address. Any value for IPC\_UID and IPC\_GID can be specified. Only the mode bits that are documented for semget argument Semaphore\_Flags can be set. This Command can only be executed by a process that has an effective user ID equal either to that of a process with appropriate privileges (see “Authorization” on page 8) or to the value of IPC\_CUID or IPC\_UID in the SEMID\_DS data structure that is associated with Semaphore\_ID. This information is taken from the buffer that is pointed to by the Argument\_address parameter. For the data structure, see “BPXYSEM – Map interprocess communication semaphores” on page 1146, SEMID\_DS DSECT.

### **Ipc\_SET\_EX**

When the AMODE of the caller is 31, it assumes the same input as IPC\_SET when AMODE is 31 and provides the same output as IPC\_SET when AMODE is 64 (that is, extended POSIX times are provided). When the AMODE of the caller is 64, IPC\_SET\_EX and IPC\_SET are equivalent.

### **Ipc\_RMID**

Removes the semaphore identifier that is specified by Semaphore\_ID from the system and destroys the set of semaphores and the SEMID\_DS data structure that are associated with it. This Command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges (see “Authorization” on page 8) or to the value of IPC\_CUID or IPC\_UID in the SEMID\_DS data structure that is associated with Semaphore\_ID.

### **SValue**

Supplied parameter

#### **Type:**

Integer

#### **Length:**

Fullword

Specifies the value to be set for the semaphore that is identified by the Semaphore\_Number.

### **Argument\_address (Buffer | Array)**

Supplied parameter

#### **Type:**

Address

#### **Length:**

Fullword

The name of a field that contains the address of the Buffer, Array or a null.

Table 28. Calling parameters and commands

Number	Command	Buffer   Array	Return Value
<b>Sem No.</b>	GETVAL	NA	SemVal, -1
<b>Sem No.</b>	SETVAL	SValue	0, -1
<b>Sem No.</b>	GETPID	NA	Pid, -1
<b>Sem No.</b>	GETNCNT	NA	Count, -1
<b>Sem No.</b>	GETZCNT	NA	Count, -1
<b>NA</b>	GETALL	Array, output	0, -1
<b>NA</b>	SETALL	Array, input	0, -1
<b>NA</b>	STAT	Buffer, output	0, -1
<b>NA</b>	SET	Buffer, input	0, -1
<b>NA</b>	RMID	NA	0, -1

**Buffer**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length of SEMID\_DS.

The name of a fullword (doubleword) field that contains the address of a data area that is mapped by SEMID\_DS. This field is used for stat and set.

**Array**

Supplied and returned parameter

**Type:**

Structure

**Length:**

GETALL - An array of 2-byte integers for each semaphore in the set equal to (SEM\_NSEMS \* 2).

SETALL - A 2-byte integer for each semaphore in the set equal to (SEM\_NSEMS \* 2).

SETVAL - A 4-byte integer for the specified semaphore. The valid range is 0 through the system limit.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semctl service returns -1, if not successful, or the following when successful:

**GETVAL**

The value of semval is returned

**GETPID**

The value of sempid is returned

**GETNCNT**

The value of semncnt is returned

**GETZCNT**

The value of semzcnt is returned

**All others**

A value of zero is returned

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semctl service stores the return code. The semctl service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The semctl service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Operation permission (read or alter) is denied to the calling process. The following reason code can accompany the return code: JRIPcDenied.
<b>EFAULT</b>	The Buffer or ARRAY parameter specified an address that caused the callable service to program check. The following reason code can accompany the return code: JRBadAddress.
<b>EINVAL</b>	One of the following errors occurred: <ul style="list-style-type: none"> <li>• Semaphore_ID is not a valid semaphore identifier.</li> <li>• Semaphore_Number is less than zero or greater than or equal to the number of semaphores in this set.</li> <li>• The Command parameter is not a valid command.</li> <li>• The mode bits were not valid (ipc_SET).</li> </ul> The following reason codes can accompany the return code: JRIPcBadFlags, JRIPcBadID, JRSema4BadSemN and JRBadEntryCode.
<b>EPERM</b>	The Command was IPC_RMID or IPC_SET, and the effective user ID of the caller is not that of a process with appropriate privileges (see “Authorization” on page 8) and is not the value of IPC_CUID or IPC_UID in the SEMID_DS data structure that is associated with Semaphore_ID. The following reason code can accompany the return code: JRIPcDenied.
<b>ERANGE</b>	The SETVAL or SETALL value exceeds the system-imposed maximum that is defined by SEM#MAX_VAL in BPXYSEM. The following reason code can accompany the return code: JRSema4BadValue.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semctl service stores the reason code. The semctl service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. Each semaphore in the semaphore set is represented by a data structure that is defined as follows:

**semval**

Unsigned halfword semaphore value

**sempid**

Process ID of the last operation

**semncnt**

Unsigned halfword number of processes waiting for semval to become greater than the current value

**semzcnt**

Unsigned halfword number of processes waiting for semval to become zero

2. The Semaphore\_ID was obtained from semget (BPX1SGT, BPX4SGT).
3. A semadj variable is maintained by the process for all of its threads. This adjustment value allows the kernel to restore semaphore values if a process terminates before it can issue a semop. It is the application's responsibility to maintain semadj values for process termination.
4. Ipc\_SET can change permissions, and might affect a thread's ability to use the semaphore functions.
5. When a semaphore ID is removed (Ipc\_RMID) from the system, all waiting threads regain control with RV=-1, RC=EIDRM, and RC=JRIPcRemoved.
6. The remove is complete by the time control is returned to the caller.

## Related services

- [“mvsprocclp \(BPX1MPC, BPX4MPC\) — Clean up kernel resources ” on page 484](#)
- [“semget \(BPX1SGT, BPX4SGT\) — Create or find a set of semaphores” on page 718](#)
- [“semop \(BPX1SOP, BPX4SOP\) — Perform semaphore serialization operations” on page 723](#)

## Characteristics and restrictions

The invoker is restricted by ownership and read and read/write permissions that are defined by semget and semctl Ipc\_SET.

## Examples

See [“BPX1SCT \(semctl\) example” on page 1284](#).

# semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores

## Function

The semget function creates a new semaphore set or finds an existing semaphore set. The semaphore set ID is returned.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SGT):</b>	31-bit
<b>AMODE (BPX4SGT):</b>	64-bit

<b>Operation</b>	<b>Environment</b>
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SGT, (Key,
               Number_of_Semaphores,
               Semaphore_Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SGT with the same parameters.

## Parameters

### Key

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Identification for this semaphore set. This is either a user-defined value that serves as a lookup value to determine if the semaphore set already exists, or the reserved value `Ipc_PRIVATE`. (See [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098. `Ipc_PRIVATE` is sometimes used when a process does not want to share a semaphore set, or when it wants to privately control access to it by other processes.)

### Number\_of\_Semaphores

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The number of semaphores that are to be allocated to this set. This value may be zero if the application knows that the semaphore set should already be created for the specified key parameter. A zero value is not allowed with `Ipc_CREAT` or `Ipc_PRIVATE`. The maximum for this variable is controlled by the installation. For an existing semaphore identifier, this variable must not be greater than the number of semaphores in that set.

### Semaphore\_Flags

Supplied parameter

#### Type:

Structure

#### Length:

Fullword

Valid values for this field include any combination of the following (additional bits cause an `EINVAL`):

**IpC\_CREAT**

Creates a message queue if the key specified does not already have an associated ID. IpC\_CREAT is ignored when IpC\_PRIVATE is specified.

**IpC\_EXCL**

Causes the semget function to fail if the key specified has an associated ID. IpC\_EXCL is ignored when IpC\_CREAT is not specified, or when IpC\_PRIVATE is specified.

**IpC\_BINSEM**

Binary semaphore. The semaphore must behave in a binary manner: the number of semaphore operations must be 1, and the semop must be either 1 with a semval of 1, or -1 with a semval of 0 or 1. Specifying the SEM\_UNDO flag in the SEM\_FLGS field of BPXYSEM on a semop() request against a binary semaphore allows the semaphore to be released when a process exits without releasing it. The use of this flag improves performance if the PLO instruction is available on the hardware.

**IpC\_SHORTHOLD**

Indicates that the application will hold the resource that is being serialized for extremely short intervals of time. When the IPC\_BINSEM flag is also specified, the default first-in-first-out ordering of semaphore obtain requesters is bypassed, allowing short duration requesters to cut to the front of the wait chain.

**S\_IRUSR**

Permits the process that owns the semaphore set to read it.

**S\_IWUSR**

Permits the process that owns the semaphore set to alter it.

**S\_IRGRP**

Permits the group that is associated with the semaphore set to read it.

**S\_IWGRP**

Permits the group that is associated with the semaphore set to alter it.

**S\_IROTH**

Permits others to read the semaphore set.

**S\_IWOTH**

Permits others to alter the semaphore set.

The values that begin with the "IpC\_" prefix are defined in BPXYIPCP and are mapped onto S\_TYPE, which is in BPXYMODE. (See [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098 and [“BPXYMODE — Map the mode constants”](#) on page 1106.)

The values that begin with the "S\_I" prefix are defined in BPXYMODE, and are a subset of the access permissions that apply to files.

This operand is ignored if the semaphore set is already defined to the system.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semget service returns the semaphore identifier or, if unsuccessful, -1.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semget service stores the return code. The semget service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The semget service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	A semaphore identifier exists for the Key parameter, but access permission, as specified by the low-order 9 bits of the Semaphore_Flags parameter, is not granted (the "S_" items). The following reason code can accompany the return code: JRIPCDenied.
<b>EEXIST</b>	A semaphore identifier exists for the Key parameter, and both Ipc_CREAT and Ipc_EXCL are specified. The following reason code can accompany the return code: JRIPCExists.
<b>EINVAL</b>	<p>Number_of_Semaphores is not valid when:</p> <ul style="list-style-type: none"> <li>• The semaphore identifier exists for the Key parameter and Number_of_Semaphores exceeds the number of semaphores previously defined.</li> <li>• Number_of_Semaphores is zero.</li> <li>• Number_of_Semaphores exceeds the system limit. This system limit is set with the IPCSEMNSEMS parameter in a BPXPRM parmlib member. You can use the <code>ipcs -x</code> shell command to view this value.</li> </ul> <p>The Semaphore_Flags parameter includes bits that are not supported by this function. The following reason codes can accompany the return code: JRSema4BadNSems, JRSema4ZeroNSems, JRSema4BigNSems, and JRIPCBadFlags.</p>
<b>ENOENT</b>	A semaphore identifier does not exist for the Key parameter and Ipc_CREAT was not set. The following reason code can accompany the return code: JRIPCNoExists.
<b>ENOSPC</b>	A semaphore identifier is to be created, but the system-imposed limit on the maximum number of allocated semaphore identifiers system-wide would be exceeded. This system limit is set with the IPCSEMNIDS parameter in the BPXPRM parmlib member. You can use <b>ipcs -x shell</b> command to view this value. You can use the <b>ipcrm</b> shell command to remove unused semaphore identifiers. The following reason code can accompany the return code: JRIPCMaxIDs.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the semget service stores the reason code. The semget service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. Each semaphore in the semaphore set is represented by a data structure that is defined as follows:

#### semval

Unsigned halfword semaphore value

**sempid**

Process ID of last operation

**semncnt**

Unsigned halfword number of processes waiting for semval to become greater than current value

**semzcnt**

Unsigned halfword number of processes waiting for semval to become zero

2. When a semaphore set is created, the value of semval for all semaphores is set to zero.
3. As long as the semaphore ID is known and access is permitted, any thread can invoke semctl or semop without invoking semget.
4. This function returns the semaphore identifier that is associated with the Key parameter.
5. When it is successful, this function creates a data structure that is defined by SEMID\_DS and an array that contains the number of semaphores specified, if one of the following is true:
  - The Key parameter is equal to Ipc\_PRIVATE.
  - The Key parameter does not already have a semaphore identifier associated with it, and Ipc\_CREAT is set.

For the data structure, see [“BPXYSEM — Map interprocess communication semaphores”](#) on page 1146.

6. Upon creation, the data structure that is associated with the new semaphore identifier is initialized as follows:
  - Ipc\_CUID and Ipc\_UID are set to the effective user ID of the calling process.
  - Ipc\_CGID and Ipc\_GID are set to the effective group ID of the calling process.
  - The low-order 9 bits of Ipc\_MODE are equal to the low-order 9 bits of the Semaphore\_Flags parameter.
  - SEM\_NSEMS is set equal to the value of the Number\_of\_Semaphores parameter.
  - SEM\_otime is set to 0 and SEM\_ctime is set to the current time.
7. If the Key parameter is not Ipc\_PRIVATE, Ipc\_EXCL is not set, and a semaphore identifier already exists for the specified Key parameter, the value of the Number\_of\_Semaphores parameter that is specified may not exceed the Number\_of\_Semaphores specified on the semget that created the semaphore set.
8. The semaphore set is removed from the system as soon as BPX1SCT/BPX4SCT (semctl RMID) is processed.
9. Users of Ipc\_PRIVATE semaphore sets are responsible for removing them when they are no longer needed. Failure to do so ties up resources.
10. Semaphores created with the Ipc\_BINSEM attribute show this bit, and may also show the Ipc\_PLOinUse bit, in the S\_MODE byte that is returned with the w\_getipc request.

**Related services**

- [“w\\_getipc \(BPX1GET, BPX4GET\) — Query interprocess communications”](#) on page 991
- [“semctl \(BPX1SCT, BPX4SCT\) — Perform semaphore control operations ”](#) on page 713
- [“semop \(BPX1SOP, BPX4SOP\) — Perform semaphore serialization operations”](#) on page 723

**Characteristics and restrictions**

- There is a maximum number of semaphore sets and semaphores that are allowed in the system.
- The invoker is restricted by ownership, read, and read-write permissions that are defined by semget and semctl Ipc\_SET.



**Examples**

See “BPX1SGT (semget) example” on page 1287.

## semop (BPX1SOP, BPX4SOP) – Perform semaphore serialization operations

---

**Function**

The semop service performs a group of semaphore operations atomically.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SOP):</b>	31-bit
<b>AMODE (BPX4SOP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SOP, (Semaphore_ID,
               Semaphore_Operations,
               Number_of_Semaphore_Operations,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SOP with the same parameters. The Semaphore\_Operations parameter is a doubleword.

**Parameters****Semaphore\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Halfword

Specifies the semaphore identifier.

**Semaphore\_Operations**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

A fullword (doubleword) that points to an array of data structures mapped by SEM\_BUF\_ELE in [“BPXYSEM — Map interprocess communication semaphores”](#) on page 1146. The SEM\_OP operations modify the semval for a specific semaphore in the semaphore set specified by SEM\_NUM. All updates to the semaphores' semval are made atomically when this callable service returns successfully. Partial updates to semval are not performed. Each SEM\_BUF\_ELE element contains the following:

- SEM\_NUM is a halfword semaphore number in the Semaphore\_ID set. References to semval, sempid, semncnt, semzcnt are to this element in the semaphore set. SEM\_NUM ranges from 0 to Number\_of\_Semaphore\_Operations - 1.
- SEM\_OP is a signed halfword with three different operations, described as follows:
  - SEM\_OP < 0, evaluate semval + SEM\_OP (remember that SEM\_OP is negative). If the operation yields a negative number, the operation either returns to the caller (EAGAIN) or suspends execution of the calling thread until the operation yields a non-negative number. Semncnt is incremented for each thread that is waiting, and decremented when waiting is complete. When waiting is complete, semval = semval + SEM\_OP.
  - SEM\_OP > 0, set semval = semval + SEM\_OP.
  - SEM\_OP = 0, test semval. If not zero, the operation either returns to the caller (EAGAIN) or suspends execution of the calling thread until semval=0. Semzcnt is incremented for each thread that is waiting, and decremented when waiting is complete.
- SEM\_FLGS – contains the Ipc\_NOWAIT and Sem\_UNDO bits. Ipc\_NOWAIT causes SEM\_OP=0 and SEM\_OP<0 to return immediately with a return code of EAGAIN if the condition cannot be met. Otherwise, processing is suspended. Sem\_UNDO instructs the process to maintain an adjustment value for SEM\_OP ^= 0. For the data structure, see [“BPXYSEM — Map interprocess communication semaphores”](#) on page 1146.

**Number\_of\_Semaphore\_Operations**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

Contains the number of operations in Semaphore\_Operations. A value of zero up to the maximum allowed by the system may be specified.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semop service returns 0 (all SEM\_OP operations were performed) or -1 (none of the SEM\_OP operations were performed).

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the semop service stores the return code. The semop service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in

*z/OS UNIX System Services Messages and Codes* The semop service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Permission is denied. The following reason code can accompany the return code: JRIPCDenied.
<b>EAGAIN</b>	The operation would result in suspension of the calling process, but NOWAIT (see SEM_FLGS) was specified. The following reason code can accompany the return code: JRIPCRetry.
<b>EDEADLK</b>	The combination of operations can never be satisfied. This condition is detected by analysis of the operations that were requested and the system maximums, and does not include interactions with other threads. For example, an operation could add 1 to a semaphore, and a later operation in the same SEM_BUF could test it for zero. The following reason code can accompany the return code: JRDeadlock.
<b>EFAULT</b>	The Semaphore_Operations parameter specified an address that caused the service to program check. The following reason code can accompany the return code: JRBadAddress.
<b>EFBIG</b>	SEM_NUM is less than zero or is greater than or equal to the number of semaphores in the set specified by the Number_of_Semaphores parameter of the semget() call. The following reason code can accompany the return code: JRSema4BadSemN.
<b>EIDRM</b>	Semaphore_ID was removed from the system while the caller was waiting. The following reason code can accompany the return code: JRIPCRemoved.
<b>EINTR</b>	semop() was interrupted by a signal. The following reason code can accompany the return code: JRIPCSignaled.
<b>EINVAL</b>	The Semaphore_ID does not represent a semaphore set. The following reason code can accompany the return code: JRIPCBadID.
<b>ENOSPC</b>	The space that is allotted for all semaphore data would be exceeded by the addition of the UNDO structure for this request. The following reason code can accompany the return code: JRSemStorageLimit.
<b>ERANGE</b>	An operation would cause sem_val or sem_adj to overflow the system-imposed limit. These system limits are defined in BPXYSEM fields SEM#MAX_VAL and SEM#MAX_ADJ. The following reason codes can accompany the return code: JRSema4BadValue and JRSema4BadAdj.
<b>E2BIG</b>	Number_of_Semaphore_Operations exceeds the maximum allowed by the system. This system limit is set with the IPCSEMNOps parameter in a BPXPRMxx parmlib member. You can use the <b>ipcs -x</b> shell command to view this value. The following reason code can accompany the return code: JRSema4BadNOps.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the semop service stores the reason code. The semop service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. Each semaphore in the semaphore set is represented by an anonymous data structure, which is defined as follows:

**semval**

Unsigned halfword semaphore value

**sempid**

Process ID of last operation

**semmcnt**

Unsigned halfword number of processes waiting for semval to become greater than current value

**semzcnt**

Unsigned halfword number of processes waiting for semval to become zero

2. A nonzero SEM\_OP value requires write permission (else EACCES).
3. A zero SEM\_OP value requires read permission (else EACCES).
4. Upon successful completion, sempid equals the process ID of the calling process.
5. Wait queue service is unpredictable.
6. Waiting is done on a thread basis. Multiple threads (even within a single process) could be waiting on the same semaphore.
7. Adjustments are maintained on a process basis, and can be changed by threads outside or within the process.
8. Within an array of semaphore operations, either all operations or none of the operations are performed.
9. Incorrect usage of semaphores may cause the application to become deadlocked and wait forever. Designing the semaphore hierarchy so that the semaphores are obtained in a specific order will avoid deadlocks.
10. If the Number\_of\_Semaphore\_Operations is zero, the callable service returns successfully with no semaphore operation being performed.

## Related services

- [“mvsprocclp \(BPX1MPC, BPX4MPC\) — Clean up kernel resources ” on page 484](#)
- [“semctl \(BPX1SCT, BPX4SCT\) — Perform semaphore control operations ” on page 713](#)
- [“semget \(BPX1SGT, BPX4SGT\) — Create or find a set of semaphores” on page 718](#)

## Characteristics and restrictions

The invoker is restricted by ownership, read, and read-write permissions that are defined by semget and semctl Ipc\_SET.

## Examples

See [“BPX1SOP \(semop\) example” on page 1292](#).

# send (BPX1SND, BPX4SND) — Send data on a socket

---

## Function

The send callable service sends data on a socket.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SND):</b>	31-bit task or SRB mode
<b>AMODE (BPX4SND):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SND,(Socket_descriptor,
               Buffer_length,
               Buffer,
               Buffer_alet,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SND with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the send is to be done.

### Buffer\_length

Supplied and returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the length of Buffer.

### Buffer

Supplied parameter

#### Type:

Character

#### Length:

Length specified by Buffer\_length

## send (BPX1SND, BPX4SND)

The name of a field that contains the data that is to be transmitted.

### Buffer\_alet

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the ALET for Buffer.

You should specify a Buffer\_alet of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### Flags

Supplied parameter

#### Type:

Structure

#### Length:

Fullword

The name of a field that contains information about how the data is to be sent. See [“BPXYMSGF — Map the message flags”](#) on page 1107 for more information about the format of this field.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the send service returns one of the following:

- The number of bytes sent from the buffer, if the request is successful. A value of 0 indicates that the connection is closed.
- -1, if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the send service stores the return code. The send service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The send service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	Socket_descriptor does not refer to a valid descriptor. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	Connection reset by peer.
<b>EINTR</b>	A signal interrupted the send before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.

Return_code	Explanation
<b>EIO</b>	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
<b>EMSGSIZE</b>	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTCONN</b>	The socket is not connected. The following reason code can accompany the return code: JRSocketNotCon.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EPIPE</b>	An attempt was made to send to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a SIGPIPE signal.
<b>EWOULDBLOCK</b>	<ul style="list-style-type: none"> <li>The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available.</li> <li>The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option.</li> </ul> <p>The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the send service stores the reason code. The send service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The socket must be connected.
2. If there is not enough room to write the data to the output buffer, the service either blocks waiting for room, or returns an EWOULDBLOCK (depending on whether the socket is marked as blocking or nonblocking, and whether SO\_SNDTIMEO timeout value was reached before space became available).
3. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

**Related services**

- [“recv \(BPX1RCV, BPX4RCV\) — Receive data on a socket and store it in a buffer” on page 680](#)

**Characteristics and restrictions**

None.

**Examples**

See “BPX1SND (send) example” on page 1291.

## send\_file (BPX1SF, BPX4SF) – Send a file on a socket

---

**Function**

The send\_file callable service sends a file, with optional header and trailer data, as a byte stream on a socket connection. The service also provides options to close the socket connection after the data is sent, and to prepare the socket for reuse after it is closed.

**Requirements****Operation****Environment****Authorization:**

Supervisor state or problem state, any PSW key

**Dispatchable unit mode:**

Task

**Cross memory mode:**

PASN = HASN

**AMODE (BPX1SF):**

31-bit

**AMODE (BPX4SF):**

64-bit

**ASC mode:**

Primary mode

**Interrupt status:**

Enabled for interrupts

**Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SF, (Sfpl_length,
              Sfpl,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SF with the same parameters. All addresses in the Sfpl structure are doublewords.

**Parameters****Sfpl\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the Sfpl structure that is being passed in the Sfpl parameter. To determine the value of Sfpl\_length, use the BPXYSFPL macro(“BPXYSFPL – Map the send\_file parameter list” on page 1147).

**Sfpl**

Supplied and returned parameter



**Type:**

Structure

**Length:**

Specified by the Sfpl\_length parameter

The name of the Sfpl structure that is to be used to control this I/O operation. See the usage notes for details on setting the fields of this structure.

The Sfpl is mapped by the BPXYSFPL macro ([“BPXYSFPL — Map the send\\_file parameter list” on page 1147](#)).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the send\_file service returns the following return codes:

- 0, if the request is successful.
- -1, if the request is not successful.
- 1, if the request was interrupted by a signal, or if a nonblocking descriptor would have blocked while sending the data. The Sfpl structure is updated by the system to account for the data that was sent. You can continue the operation from the point at which it was interrupted by reissuing the send\_file request with the same Sfpl structure.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the send\_file service stores the return code. The send\_file service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The send\_file service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	A descriptor is marked nonblocking, and no data could be sent without blocking.
<b>EBADF</b>	A descriptor that was not valid was supplied; the file was not open for reading; or the socket was not open for writing. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen, JRRFileWrOnly, JRWFileRdOnly.
<b>ECONNRESET</b>	The connection was reset by a peer. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRSockNotCon.
<b>ECONNABORTED</b>	A connection was dropped.
<b>EFAULT</b>	An address that was passed could not be referenced in the key of the caller.
<b>EIO</b>	An I/O error occurred.

Return_code	Explanation
<b>ENOBUFS</b>	The service was unable to obtain a buffer. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOMEM</b>	The service was unable to obtain memory to complete the operation.
<b>EINTR</b>	A signal interrupted the send_file service before any data was written. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	Data that was not valid was sent to the request. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRSocketCallParmError.
<b>ENOTCONN</b>	The socket was not connected. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRSocketNotCon.
<b>EPIPE</b>	An attempt was made to send a message to a socket that is shut down or closed. This error also generates a SIGPIPE signal. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRSocketClosed.
<b>EWOULDBLOCK</b>	A descriptor is marked nonblocking and no data could be sent, or the SO_SNDTIMEO timeout value was reached before space became available.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the send\_file service stores the reason code. The send\_file service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes****1. Sfpl Structure (send\_file parameter list)**

The send\_file operation is controlled by the values that are set into this structure. Refer to the BPXYSFPL macro for the exact field names ([“BPXYSFPL — Map the send\\_file parameter list”](#) on page 1147).

**Note:**

- The entire Sfpl structure should be zeroed out before its first use, to ensure that undefined options, reserved space, and fields that might be used are initialized properly.
- All doubleword fields are treated as signed 63-bit arithmetic values. For operations that are known to be under 4 gigabytes in size, you can refer to the lower words of these fields (named in the BPXYSFPL macro) as unsigned 32-bit arithmetic values.

**Field****Description****Socket\_desc**

The descriptor on which to send the data.

**File\_desc**

The descriptor from which to read the data that is sent.

**File\_offset**

A doubleword field that contains the byte offset in the file from which to start sending.

**File\_bytes**

A doubleword field that contains the number of bytes to be sent, starting from the File\_offset.

If this field is -1, the entire file, from File\_offset, is sent. The system updates the field with the number of file bytes that were sent (File\_size-File\_offset).

If this field is 0, no file data is sent, and File\_desc is ignored.

**File\_size**

A doubleword field that is updated by the system after the operation with the file's size.

**Header\_len**

The length of the header data.

**Header\_ptr**

The address of the header data that is to be sent in front of the file data.

**Header\_alet**

The ALET of the header data.

**Trailer\_len**

The length of the trailer data.

**Trailer\_ptr**

The address of the trailer data that is to be sent after the file data.

**Trailer\_alet**

The ALET of the trailer data.

**Bytes\_sent**

A doubleword field that is filled in by the system with the total number of bytes that are sent on this call. If the file must be sent with multiple calls because of signal interruptions, this field contains the value for the last call only; it is not a running total.

**Options**

A field that contains the following bit flags, which have the specified meaning when the bit is on:

- SF\_CLOSE — Close the Socket\_desc after the data has been sent. If the operation completes successfully and Socket\_desc is closed, the system updates Socket\_desc in the Sfpl with -1.
- SF\_REUSE — Prepare the Socket\_desc to be reused after the data has been successfully sent.

This option is intended for sockets, and for the subsequent use of the descriptor on an accept\_and\_recv() call. To reuse the socket descriptor, the Socket\_desc value, as updated by the system in the Sfpl after the call to send\_file(), is specified as the Accepted\_socket parameter on the accept\_and\_recv() call.

Between the send\_file() and the accept\_and\_recv() calls, a reused socket may only be used on accept\_and\_recv() or close(). The socket descriptor should be closed if it is not to be used again.

If reuse is not supported, the system closes Socket\_desc, and replaces its value in the Sfpl with -1. This ensures that the output value of Socket\_desc is always appropriate as an input value for the Accepted\_socket parameter of an accept\_and\_recv() call.

The SF\_REUSE option is not supported.

2. The send\_file() function attempts to send the header data, followed by the file data from File\_desc, followed by the trailer data, over socket\_desc.
3. As data is sent, the system updates the Sfpl structure to account for the data that has been sent. This facilitates continuation after a signal interruption, but it also means that the Sfpl must be almost completely reset to start another new operation.
4. If File\_offset>File\_size, or File\_bytes>(File\_size-File\_offset), the operation fails with an EINVAL error.

5. The SF\_CLOSE and SF\_REUSE flags are only effective when the operation completes successfully.
6. The send\_file service is not strictly limited to sending a file on a socket. Any two stream-oriented descriptors may be used, although some of the parameters may have to be interpreted differently. When File\_desc is a pipe, for example, the File\_size and Offset parameters are meaningless.
7. The file cursor for the File\_desc that is specified is updated with the results of the send\_file operation. This does not affect other send\_file() calls, but it does affect later read() and write() operations that use this File\_desc.

**8. Application Usage**

The send\_file service is designed to work with the accept\_and\_recv service to provide an efficient file transfer capability for a connection-oriented server with short connection times and high connection rates.

These functions are designed for a server process/thread model that is different from the traditional one in which a parent thread accepts connections in a loop and spins off child processes or threads to issue the receive and do work. In this new server model, the parent is eliminated. Multiple worker processes or threads are initially created, and each worker process or thread independently executes the accept\_and\_recv() and send\_file() functions in a loop.

The performance benefits of accept\_and\_recv() and send\_file() over the separate operations that they combine include fewer buffer copies, recycled sockets, and optimal thread scheduling.

Socket descriptors can be recycled in the following way:

- a. On the first call to accept\_and\_recv(), the application sets the Accepted\_socket parameter to -1. This causes the system to assign a new descriptor to the accepted socket.
- b. On the following call to send\_file(), the application requests SF\_REUSE. The socket session is closed, but the socket descriptor remains available for reuse on the next accept\_and\_recv().
- c. All later calls to accept\_and\_recv() specify as their Accepted\_socket the Socket\_desc value that is left over from the previous call to send\_file().

In cases in which the socket does not support reuse, the system sets Socket\_desc to -1 after the send\_file(), so that the value is suitable for the next accept\_and\_recv() call.

**Related services**

- [“accept\\_and\\_recv \(BPX1ANR, BPX4ANR\) — Accept a connection and receive the first block of data” on page 19](#)
- [“send \(BPX1SND, BPX4SND\) — Send data on a socket ” on page 726](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1SF \(send\\_file\) example” on page 1286](#).

## **sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket**

---

**Function**

The sendmsg callable service sends messages on a socket.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX2SMS):</b>	31-bit
<b>AMODE (BPX4SMS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX2SMS,(Socket_descriptor,
               Message_hdr,
               Flags,
               Iov_alet,
               Iov_buffer_alet,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SMS with the same parameters. All addresses in the Message\_hdr structure are doublewords.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the sendmsg is to be done.

### Message\_hdr

Supplied parameter

#### Type:

Structure

#### Length:

Length of BPXYMSGH

The name of a field that contains the message header, which describes how the message is to be sent. In 64-bit mode, Message\_hdr contains doubleword pointer subfields, and points to an Iov\_struct structure that contains doubleword pointer and length subfields. See [“BPXYMSGH — Map the message header” on page 1109](#) for more information about the format of this field.

### Flags

Supplied parameter

**Type:**  
Structure

**Length:**  
Fullword

The name of a field that contains information about how the data is to be sent. See [“BPXYMSGF — Map the message flags”](#) on page 1107 for more information about the format of this field.

**Iov\_alet**  
Supplied parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a field that contains the ALET for the IOV structure that is specified in Message\_hdr.

**Iov\_buffer\_alet**  
Supplied parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a field that contains the ALET for the buffers that are pointed to by the IOV structure that is specified in Message\_hdr.

**Return\_value**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the sendmsg service returns one of the following:

- The number of bytes sent from the buffers, if the request is successful.
- -1, if the request is not successful.

**Return\_code**  
Returned parameter

**Type:**  
Integer

**Length:**  
Fullword

The name of a fullword in which the sendmsg service stores the return code. The sendmsg service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sendmsg service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EACCES	For AF_UNIX sockets, search permission is denied for a component of the path prefix or write access to the named socket is denied.
EAFNOSUPPORT	The address family that was specified in the message header is not the same as the address family that owns the socket.

Return_code	Explanation
<b>EBADF</b>	A file descriptor that was not valid was supplied. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	Connection reset by peer. The following reason code can accompany the return code: JRSockNotCon.
<b>EINTR</b>	A signal interrupted the sendmsg service before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	Data that was not valid was sent to the request. The following reason codes can accompany the return code: JRInvalidMsggh, JRSocketCallParmError, and JRSockNoName.
<b>EIO</b>	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
<b>EMSGSIZE</b>	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTCONN</b>	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EPIPE</b>	An attempt was made to send a message to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a SIGPIPE signal.
<b>EWouldBLOCK</b>	<ul style="list-style-type: none"> <li>The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available.</li> <li>The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option.</li> </ul> <p>The following reason codes can accompany the return code: JRWouldBlock and JRTimeout.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sendmsg service stores the reason code. The sendmsg service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. The socket can be either connected or unconnected.
2. If there is not enough room to write the data to an output buffer, the service either blocks waiting for an output buffer to become available, or returns an EWouldBLOCK (depending on whether the socket

## sendto (BPX1STO, BPX4STO)

is marked as blocking or nonblocking, and whether SO\_SNDTIMEO timeout value was reached before space became available).

3. The BPX2SMS/BPX4SMS call supersedes the BPX1SMS call, which is still available for migration purposes only.
4. If the recvmmsg security label is not equivalent to the sendmsg security label when access rights are passed on the sendmsg, the new descriptors are not created.
5. The number of buffers that are pointed to by the IOV structure in Message\_hdr may not exceed IOV\_MAX (defined in “BPXYIOV — Map the I/O vector structure” on page 1098).
6. See Chapter 3, “Callable services for SRB mode routines,” on page 1041 for more information about programming considerations for SRB mode.

### Related services

“recvmmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers” on page 687

### Characteristics and restrictions

None.

### Examples

See “BPX2SMS (sendmsg) example” on page 1290.

## sendto (BPX1STO, BPX4STO) — Send data on a socket

---

### Function

The sendto callable service sends data on a socket.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1STO):</b>	31-bit task or SRB mode
<b>AMODE (BPX4STO):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1STO,(Socket_descriptor,  
              Buffer_length,  
              Buffer,  
              Buffer_alet,  
              Flags,
```



```

Sockaddr_length,
Sockaddr,
Return_value,
Return_code,
Reason_code)

```

AMODE 64 callers use BPX4STO with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the sendto is to be done.

### Buffer\_length

Supplied and returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the length of Buffer.

### Buffer

Supplied parameter

#### Type:

Character

#### Length:

Length specified by Buffer\_length

The name of a field from which the data is to be sent.

### Buffer\_alet

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the ALET for Buffer. You should specify a Buffer\_alet of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### Flags

Supplied parameter

#### Type:

Structure

#### Length:

Fullword

The name of a field that contains information about how the data is to be sent. See [“BPXYMSGF — Map the message flags” on page 1107](#) for more information about the format of this field.

**Sockaddr\_length**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the length of Sockaddr. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer that is specified should be the maximum length that the sockaddr could be on output.

**Sockaddr**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length specified by Sockaddr\_length

The name of a socket address structure to which the data is to be sent. For more information about the format of this field, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sendto service returns one of the following:

- The number of bytes that were sent on the socket, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sendto service stores the return code. The sendto service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The sendto service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	For AF_UNIX sockets, search permission is denied for a component of the path prefix or write access to the named socket is denied.
<b>EAFNOSUPPORT</b>	The address family that was specified in the sockaddr is not the same address family as the socket.
<b>EBADF</b>	A file descriptor that was not valid was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	Connection reset by peer. The following reason code can accompany the return code: JRSocketNotCon.

Return_code	Explanation
<b>EINTR</b>	A signal interrupted the sendto service before any data was written. The following reason code can accompany the return code: JRSocketRdwrSignal.
<b>EINVAL</b>	An input parameter was incorrect. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoName.
<b>EIO</b>	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JrPrevSockError.
<b>EMSGSIZE</b>	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTCONN</b>	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EPIPE</b>	An attempt was made to send to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a SIGPIPE signal.
<b>EPROTOTYPE</b>	The address specifies a socket that is not the correct type for this request.
<b>EWouldBlock</b>	<p>The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available.</p> <p>The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option.</p> <p>The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sendto service stores the reason code. The sendto service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. A datagram socket can be unconnected.
2. If the sending socket has no space to hold the message that is to be transmitted, the sendto service either blocks waiting for an output buffer to become available, or returns an EWouldBlock (depending on whether the socket is marked as blocking or nonblocking, and whether SO\_SNDTIMEO timeout value was reached before space became available).
3. See [Chapter 3, "Callable services for SRB mode routines," on page 1041](#) for more information about programming considerations for SRB mode.

**Related services**

- [“recvfrom \(BPX1RFM, BPX4RFM\) — Receive data from a socket and store it in a buffer” on page 683](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1STO \(sendto\) example” on page 1298](#).

## **server\_init (BPX1SIN, BPX4SIN) — Server initialization**

---

**Function**

The `server_init` callable service allows a server address space to connect to Work Load Manager (WLM) for the purpose of queueing and servicing work requests.

**Requirements****Operation****Environment****Authorization:**

Supervisor state or problem state, any PSW key

**Dispatchable unit mode:**

Task or SRB

**Cross memory mode:**

PASN = HASN

**AMODE (BPX1SIN):**

31-bit task or SRB mode

**AMODE (BPX4SIN):**

64-bit task mode only

**ASC mode:**

Primary mode

**Interrupt status:**

Enabled for interrupts

**Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SIN, (ManagerType,
               SubSystemType,
               SubSystemName,
               ApplEnv,
               ParallelEu,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SIN with the same parameters.

**Parameters****ManagerType**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains one or more of the following values that indicate the type of WLM manager the caller is requesting to become. The following are the supported values:

**SRV\_WORKMGR**

WLM work management services are to be made available to the calling address space. This value can be combined with the SRV\_QUEUEMGR and SRV\_SERVERMGR values.

**SRV\_QUEUEMGR**

WLM queue management services are to be made available to the calling address space. This value can be combined with the SRV\_WORKMGR and SRV\_SERVERMGR values.

**SRV\_SERVERMGR**

WLM server management services that are associated with a queue manager are to be made available to the calling address space. This value can be combined with the SRV\_QUEUEMGR and SRV\_WORKMGR values.

These constants are defined in the BPXYCONS macro; see [“BPXYCONS — Constants used by services”](#) on page 1062.

**SubSystemType**

Supplied parameter

**Type:**

Character string

**Length:**

4 bytes

The name of a 4-byte field that contains the generic subsystem type (such as CICS®, IMS, and WEB). When SRV\_WORKMGR is specified for the ManagerType parameter, this is the primary category under which WLM classification rules are grouped. This parameter must be padded with blanks if the name is less than 4 bytes. When SRV\_QUEUEMGR is specified for the ManagerType parameter, the combination of the SubSystemType and SubSystemName parameter values must be unique to a single MVS system.

**SubSystemName**

Supplied parameter

**Type:**

Character string

**Length:**

8 bytes

The name of an 8-byte field that contains the subsystem name to be used for classifying work requests when SRV\_WORKMGR is specified for the ManagerType parameter. This parameter must be padded with blanks if the name is less than 8 bytes. When SRV\_SERVERMGR is specified for the ManagerType parameter, the subsystem name must match the subsystem name that is specified on the corresponding call to server\_init for a work manager (ManagerType = SRV\_WORKMGR).

**ApplEnv**

Supplied parameter

**Type:**

Character string

**Length:**

32 bytes

The name of a 32-byte area that contains the name of the application environment under which work requests are served. The character string must be padded with blanks if the name is less than 32 characters. This parameter is only valid when SRV\_SERVERMGR is specified for the ManagerType parameter. It is ignored for all other ManagerType values.

**ParallelEu**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the maximum number of tasks within the address space that will be created to process concurrent work requests. This parameter is only valid when SRV\_SERVERMGR is specified for the ManagerType parameter. It is ignored for all other ManagerType values.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the server\_init service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the server\_init service stores the return code. The server\_init service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The server\_init service can return one of the following values in the Return\_code parameter:

Return code	Explanation
<b>EFAULT</b>	An argument of this service contained an address that was not accessible to the caller.
<b>EINVAL</b>	The ManagerType parameter contains a value that is not correct.
<b>EMVSWLMERROR</b>	A WLM service failed. Consult Reason_code to determine the WLM service that failed and the reason the error occurred.
<b>EPERM</b>	The calling thread's address space is not permitted to the BPX.WLMSEVER profile in the FACILITY class. The caller's address space must be permitted to the BPX.WLMSEVER profile in the FACILITY class. If the BPX.WLMSEVER resource profile is not defined, the calling process is not defined as a superuser (UID=0).
<b>EMVSSAF2ERR</b>	An error occurred in the security product. Consult Reason_code to determine the exact reason the error occurred.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the server\_init service stores the reason code. The server\_init service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.
2. A successful call to server\_init causes the calling address space to be connected to WLM for the WLM management services requested. Additionally, for a successful server manager connection call (SRV\_SERVER\_MGR ManagerType), the calling process is made a child of and placed in the session and process group of the corresponding work manager. The corresponding work manager is the process that called server\_init for the ManagerType combination SRV\_WORK\_MGR+SRV\_QUEUE\_MGR with the same SubSystemType and SubSystemName values that were specified by the server manager process.

This parent child relationship facilitates the use of signals between the server manager and the work manager to communicate with each other. The server manager, for example, after calling this service can issue the getppid service call to obtain the work server's process id, and then send signals to the work server when necessary.

Because the server manager processes are child processes of the work manager/queue manager process, the work manager/queue manager process needs to ensure that terminated server manager processes get cleaned up. This requires the parent to either prevent the children processes from becoming zombie processes by using the sigaction service for the SIGCHLD signal, or clean up any terminated child processes by using the wait service.

3. This service should be used by a server that is designed to function in one of the following two ways:

The server is divided into multiple address spaces, with a work and queue manager (MANAGER\_TYPE=SRV\_WORK\_MGR+SRV\_QUEUE\_MGR) address space obtaining work requests from an external source and then queueing the work requests to one or more server manager (MANAGER\_TYPE=SRV\_SERVER\_MGR) address spaces to process the work requests.

A single server address space functions as the work and queue manager and as the server manager (MANAGER\_TYPE=SRV\_WORK\_MGR+SRV\_QUEUE\_MGR+SRV\_SERVER\_MGR), with one or more threads obtaining work from an external source and then queueing the work requests to one or more server threads that process the work.

The first method is the recommended approach to using this service, since it takes best advantage of WLM's system workload balancing capabilities by allowing WLM to create and manage the server address spaces against all other work in the system.

4. The server\_init service is a privileged service that requires the caller to be authorized in one of the following ways:
  - Have read access to the BPX.WLMSEVER resource profile in the FACILITY class
  - Have a UID of 0 when the BPX.WLMSEVER profile is not defined

## Related services

- [“server\\_pwu \(BPX1SPW, BPX4SPW\) — Server process work unit ” on page 746](#)

## Examples

See [“BPX1SIN \(server\\_init\) example” on page 1288](#).

## server\_pwu (BPX1SPW, BPX4SPW) – Server process work unit

### Function

The server\_pwu callable service provides a general purpose interface for managing and processing work using the Work Load Manager (WLM). It lets a program put work requests onto the WLM work queues, obtain work from the WLM work queues, transfer work to other WLM work servers, end units of work, delete WLM enclaves, and refresh WLM work servers.

### Requirements

#### Operation

#### Environment

#### Authorization:

Supervisor state or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1SPW):

31-bit

#### AMODE (BPX4SPW):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1SPW, (FcnCode,
               TransClass,
               ApplEnv,
               ClassifyAreaLen,
               ClassifyAreaPtr,
               ApplDataLen,
               ApplDataPtr,
               FdStrucPtr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SPW with the same parameters.

### Parameters

#### FcnCode

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains one or more of the following values indicating the function that is requested. The following are the supported values:

#### SRV\_PUT\_NEWWRK

A new work request is to be put onto the WLM work queue for the application environment that is identified by the ApplEnv parameter, as part of a newly created WLM enclave. This value cannot be combined with any other FcnCode value.



**SRV\_PUT\_SUBWRK**

A new work request is to be put onto the WLM work queue for the application environment that is identified by the ApplEnv parameter, as part of the existing WLM enclave that is associated with the calling thread. This value can be combined only with the SRV\_END\_WRK FcnCode value.

**SRV\_TRANSFER\_WRK**

The work request that is associated with the WLM enclave of the calling thread is to be transferred to the work queue of the target application environment that is identified by the ApplEnv parameter. As part of the transfer, the calling thread is disassociated from its WLM Enclave. This value cannot be combined with any other FcnCode value.

**SRV\_GET\_WRK**

A new work request is to be obtained from the WLM work queue for the calling application environment server. The SRV\_GET\_WRK FcnCode also results in the association of the calling thread with the WLM enclave that was created when the obtained work request was put onto a WLM work queue. If the calling thread is already associated with a WLM enclave, an implicit SRV\_END\_WRK is performed. This value can only be combined with the SRV\_END\_WRK and SRV\_DEL\_ENC FcnCode values.

**SRV\_REFRESH\_WRK**

The servers that are associated with the application environments that are managed by the calling work and queue manager are to be refreshed. This causes all servers to complete existing work requests and then terminate. New servers are then started to process new work.

**SRV\_END\_WRK**

The calling thread is to be disassociated from its WLM enclave. This value can only be combined with the SRV\_GET\_WRK, SRV\_PUT\_SUBWRK and SRV\_DEL\_ENC FcnCode values.

**SRV\_DEL\_ENC**

The WLM enclave that is associated with the calling thread is to be deleted. This value can only be combined with the SRV\_GET\_WRK and SRV\_END\_WRK FcnCode values. This value should not be used to delete an enclave before ending the work units in the enclave, to prevent erroneous workload management results

**SRV\_DISCONNECT**

The calling server's connection to WLM is to be severed. Once a server is disconnected from WLM, it can no longer use this service to process more requests for the application environment for which it had been connected to WLM by a call to the server\_init function. If a SRV\_DISCONNECT is performed by a work and queue manager, all related server managers implicitly lose their connections to WLM. This means that the related server managers also lose their ability to process more requests via this service.

**SRV\_DISCONNECT\_COND**

The calling server's connection to WLM is to be severed only if the caller has no more WLM enclaves that it is still managing. A work and queue manager is still managing an enclave if it has yet to be serviced by a server manager. Once a server is disconnected from WLM, it can no longer use this service to process more requests for the application environment for which it had been connected to WLM by a call to the server\_init function. If a SRV\_DISCONNECT is performed by a work and queue manager, all related server managers implicitly lose their connection to WLM as well. This means that the related server managers also lose their ability to process more requests via this service.

These constants are defined in the BPXYCONS macro ([“BPXYCONS – Constants used by services” on page 1062](#)).

**TransClass**

Supplied parameter

**Type:**

Character string

**Length:**

8 bytes

The name of an 8-byte area that contains the name of the transaction class that is to be associated with the work request. This parameter is only valid when the SRV\_PUT\_NEWWRK FcnCode parameter value is specified. It is ignored for the other FcnCode parameter values, and ignored if a classification area is supplied. This parameter must be padded with blanks if the name contains fewer than 8 bytes.

**ApplEnv**

Supplied parameter

**Type:**

Character string

**Length:**

32 bytes

The name of a 32-byte area that contains the name of the application environment under which work requests are served. The character string must be padded with blanks, if the name contains fewer than 32 characters. This parameter is valid only when one of the SRV\_PUT FcnCode parameter values is specified, or if the SRV\_TRANSFER\_WRK function code parameter value is specified and is ignored otherwise.

**ClassifyAreaLen**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the classification information area, as specified by the ClassifyAreaPtr parameter. This parameter is valid only with the SRV\_PUT\_NEWWRK FcnCode parameter value, and is ignored for the other FcnCode parameter values.

**ClassifyAreaPtr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword

The name of a fullword that contains the address of the classification information for the work request in the form of the parameter list for the IWMCLSFY macro. This parameter is intended for use with the SRV\_PUT\_NEWWRK FcnCode parameter value only. The length of this area is supplied by the ClassifyAreaLen parameter. This parameter is ignored if the ClassifyAreaLen parameter contains a zero value. Some of the classification data that is pointed to by the IWMCLSFY parameter list is truncated if it exceeds the maximum supported length, as follows:

**ACCTINFO**

143 bytes maximum length

**SUBSYSPM**

255 bytes maximum length

**SOURCELU**

17 bytes maximum length

**COLLECTION**

18 bytes maximum length

**CORRELATION**

12 bytes maximum length

**ApplDataLen**

Supplied or returned parameter

**Type:**

Integer

**Length:**

Fullword

When one of the SRV\_PUT or SRV\_TRANSFER FcnCode parameter values is specified, this is a supplied parameter that is the name of a fullword that contains the length of the application data that is specified by the ApplDataPtr parameter.

When the SRV\_GET\_WRK FcnCode value is specified, this is a returned parameter that is the name of a fullword in which the server\_pwu service is to return the length of the application data that is returned in the ApplDataPtr parameter.

This parameter is intended for use when one of the SRV\_PUT, SRV\_TRANSFER, or SRV\_GET FcnCode parameter values is specified. The maximum length that is supported for the application data is 10 megabytes.

**ApplDataPtr**

Supplied or returned parameter

**Type:**

Address

**Length:**

Fullword

When one of the SRV\_PUT or SRV\_TRANSFER FcnCode parameter values is specified, this is a supplied parameter that is the name of a fullword that contains the address of the application data area that is to be associated with the work request. This application data allows the caller to uniquely identify the specific work that the caller is requesting.

When the SRV\_GET\_WRK FcnCode value is specified, this is a returned parameter that is the name of a fullword in which the server\_pwu service is to return the address of the application data that is associated with the obtained work request. The returned data area is an identical copy of the data area that was supplied on the corresponding server\_pwu call to put the work request on a WLM work queue.

This parameter is intended for use when one of the SRV\_PUT, SRV\_TRANSFER, or SRV\_GET FcnCode parameter values is specified.

**FdStrucPtr**

Supplied or returned parameter

**Type:**

Address

**Length:**

Fullword

When one of the SRV\_PUT or SRV\_TRANSFER FcnCode parameter values is specified, this is a supplied parameter that is the name of a fullword that contains the address of the file descriptor list structure, as mapped by the BPXYSFDL mapping macro. The file descriptors that are specified in the list are to be propagated to the process that calls the server\_pwu service to obtain the work request that is created by the call to this service. If the SFDLCLOSE flag is turned on in the SFDLFLAGS field of the supplied structure, all file descriptors in the list are closed in the calling process. If a null address is specified, no file descriptors are propagated.

When the SRV\_GET\_WRK FcnCode value is specified, this is a returned parameter that is the name of a fullword in which the server\_pwu service is to return the address of the file descriptor list structure that is associated with the obtained work request. The returned file descriptor list structure contains a count of entries and a list of file descriptors that represent the list of file descriptors that have been remapped in the calling process for the obtained work request. The remapped file descriptor values correspond to the file descriptor values that were supplied on the server\_pwu call to put the work request on a WLM work queue. A file descriptor list is only returned for a SRV\_GET\_WRK call if the list that was supplied on the corresponding SRV\_PUT\_WRK, SRV\_PUT\_SUBWRK or SRV\_TRANSFER\_WRK call had a file descriptor count of greater than zero.

This parameter is intended for use when one of the SRV\_PUT, SRV\_TRANSFER, or SRV\_GET\_WRK FcnCode parameter values is specified. The maximum number of file descriptors that are supported in the file descriptor list is 64.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the server\_pwu service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the server\_pwu service stores the return code. The server\_pwu service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The server\_pwu service can return one of the following values in the Return\_code parameter:

<b>Return code</b>	<b>Explanation</b>
<b>EAGAIN</b>	The requested service could not be performed at the current time. The following reason code can accompany this return code: JRENCLAVESEXIST
<b>EINVAL</b>	The FcnCode parameter contains a value that is not correct.
<b>EMVSWLMERROR</b>	A WLM service failed. Consult Reason_code to determine the WLM service that failed and the reason the error occurred. See <i>z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO</i> for a list of error reason codes for the WLM services (IWM*).

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the server\_pwu service stores the reason code. The server\_pwu service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Usage of the various server\_pwu functions requires that the calling address space has successfully issued a call to the server\_init service.
2. For the SRV\_PUT\_NEWWRK and SRV\_REFRESH\_WRK functions to run successfully, the caller must have successfully issued a call to the server\_init service for one of the following ManagerType parameter combinations:
  - SRV\_WORK\_MGR + SRV\_QUEUE\_MGR

- SRV\_WORK\_MGR + SRV\_QUEUE\_MGR + SRV\_SERVER\_MGR
3. For the SRV\_PUT\_SUBWRK and SRV\_TRANSFER\_WRK functions to run successfully, the caller must have successfully issued a call to the server\_init service for one of the following ManagerType parameter combinations:
    - SRV\_WORK\_MGR + SRV\_QUEUE\_MGR + SRV\_SERVER\_MGR
    - SRV\_SERVER\_MGR
  4. For the SRV\_GET\_WRK, SRV\_END\_WRK and SRV\_DEL\_ENC functions to run successfully, the caller must have successfully issued a call to the server\_init service for one of the following ManagerType parameter combinations:
    - SRV\_WORK\_MGR + SRV\_QUEUE\_MGR + SRV\_SERVER\_MGR
    - SRV\_SERVER\_MGR
  5. A successful call to server\_pwu for the SRV\_PUT\_NEWWRK FcnCode not only creates a work request that is placed onto a WLM work queue, but it also creates a new WLM enclave for that work to run in when the work request is obtained. The newly created WLM enclave is classified based on the classification information that is supplied in the input classify area, or based on the input transaction class. Unlike SRV\_PUT\_NEWWRK, the SRV\_PUT\_SUBWRK and SRV\_TRANSFER\_WRK FcnCodes queue work requests that eventually are associated with the WLM enclave of the calling thread when the work request is obtained.
  6. A successful call to server\_pwu for the SRV\_GET\_WRK FcnCode not only results in the caller's obtaining a work request from a WLM work queue that is associated with the caller's application environment, but it also results in the associating of the calling thread with the WLM enclave that is associated with the obtained work request. When the calling thread goes through task termination, or when its process is terminated, the work request is ended and the associated WLM enclave is deleted if it is owned by the terminating task or process. The SRV\_GET\_WRK caller owns the enclave if the work was queued using the SRV\_PUT\_NEWWRK or SRV\_TRANSFER\_WRK functions. If the caller is a thread created using pthread\_create (pthread), the thread task owns the enclave. If the caller is not a pthread, the process owns the enclave.

When the FdStrucPtr parameter is used to propagate file descriptors, the caller must ensure that all of the file descriptors in the list are valid open file descriptors in the caller's process and are not being closed during the processing of this service. If this is not the case, this service cannot guarantee the proper propagation of the specified file descriptors.

7. The following demonstrates some sample usage scenarios for the FdStrucPtr parameter:

- The queue manager process puts work on a work queue for a single-threaded server manager with no open file descriptors. The queued work has a supplied file descriptor structure with 3 file descriptors specified:

Fds Supplied on SRV_PUT_NEWWRK	Fds returned on SRV_GET_WRK
-----	-----
0, 1, 2	0, 1, 2

- The queue manager process puts work on a work queue for a multithreaded server manager with open file descriptors. The queued work has a supplied file descriptor structure with 3 file descriptors specified:

Fds Supplied on SRV_PUT_NEWWRK	Fds returned on SRV_GET_WRK
-----	-----
0, 1, 2	12, 9, 14

## Related services

[“server\\_init \(BPX1SIN, BPX4SIN\) – Server initialization” on page 742](#)

## Examples

See [“BPX1SPW \(server\\_pwu\) example” on page 1294](#).

# set\_dub\_default (BPX1SDD, BPX4SDD) – Set the dub default service

## Function

The set\_dub\_default service allows the calling address space to change the current default dub setting for tasks within the address space.

## Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1SDD)	31-bit
AMODE (BPX4SDD)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SDD, (Dub_setting,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SDD with the same parameters.

## Parameters

**Dub\_setting**  
Supplied parameter

**Type**  
Integer

**Length**  
Fullword

The name of a fullword that contains the dub setting value.

Specifies the name of a fullword field that must contain one of the following dub setting values:

**DUBPROCESS**  
Dub the subtasks of the caller as new processes when each issues its first z/OS UNIX service call.

**DUBTHREAD**  
Dub the subtasks of the caller as threads in the caller's process when each issues its first z/OS UNIX service call.

**DUBTASKACEE**

Dub each subtask of the caller with its own z/OS UNIX security environment, if the subtask has a task-level security environment (ACEE) associated with it.

**DUBNOSIGNALS**

Dub the caller as a process to which signals are not delivered.

**DUBPROCESSDEFER**

Dub each subtask of the caller as a new process when it issues its first z/OS UNIX service call. The address space is not dubbed when this call is issued. The first dub of the address space occurs when the next z/OS UNIX service call is issued (by this task or by another task in the address space).

**DUBJOBPERM**

Makes the entire job permanent. All processes that are dubbed in this job are considered permanent processes. A *permanent process* is a process that is not taken down during a z/OS UNIX shutdown. All z/OS UNIX callable services that are called from these processes during a shutdown and restart window will return in failure.

**DUBABENDCALLS**

All z/OS UNIX callable services that are called from a process that is registered as permanent during a shutdown and restart window will end abnormally. This option is only relevant when it is accompanied by the DUBJOBPERM option.

**DUBNOJSTUNDUB**

Does not undub the entire job step when the last dubbed task (other than the job step task) undubs.

**DUBUNIQUEACEE**

Indicates to the kernel that this address space does not share ACEEs between tasks for the life of the process.

**DUBFAILNOTREADY**

Any syscall that attempts to dub the caller as a process during a period when the z/OS UNIX kernel is shut down will result in a failing return code, EMVSINITIAL, and reason code JrKernelReady. If the syscall does not have a return code parameter, it will receive an EC6 retryable abend with reason code JNotUp.

See [“BPXYCONS — Constants used by services” on page 1062](#) for the dub setting values.

**Return\_value**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the set\_dub\_default service returns:

**0**

- If the calling task is a dubbed thread.
- If the calling task is not dubbed and the request did not include the DubProcessDefer option.

**1**

- If the calling task is a dubbed process.
- If the calling task is not dubbed and the request included the DubProcessDefer option.

**-1**

If the call is unsuccessful.

**Return\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the set\_dub\_default service stores the return code. The set\_dub\_default service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The set\_dub\_default service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EINVAL**

One of the parameters contains an unsupported or incorrect value. The following reason code can accompany the return code: JRDubSetting.

**EPERM**

The calling process does not have the appropriate privilege to perform the requested operation. The following reason code can accompany the return code: JROK.

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the set\_dub\_default service stores the reason code. The set\_dub\_default service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Summary of option behavior and requirements**

Table 29. Options and requirements for set_dub_default				
Option	Function	Defers dubbing	Issue from job step task (JST)	Dub required
DUBPROCESS	All subtasks of this task are dubbed as processes.	No	No	Yes
DUBTHREAD	All subtasks of this task are dubbed as threads. Dubbing threads is the default behavior of dub processing. With this option, applications can switch back from dubbing subtasks as processes to dubbing tasks as threads.	No	No	Yes



Table 29. Options and requirements for set\_dub\_default (continued)

Option	Function	Defers dubbing	Issue from job step task (JST)	Dub required
DUBTASKACEE	<p>This task and all subtasks of the caller are given its own z/OS UNIX security environment if the task has a security environment that is different than the address space (TCBSENV ^= ASCBSENV). Using this option, applications can create a process with a different user identity without incurring the performance overhead and storage constraints of fork() or spawn() with <i>userid</i>.</p> <p>DUBTASKACEE is the default behavior of dub processing. If a task has a TCBSENV that is different from the address space, then that task is dubbed with the task level security, regardless of the DUBTASKACEE setting.</p>	No	No	No
DUBNOSIGNALS	<p>This task will not be interrupted for signal delivery. If the job step task was blind-dubbed, it is marked as not being able to receive signals. Any application that has a task that needs to remain active and cannot tolerate a terminating signal such as a SIGKILL can use this option.</p>	No	No	No
DUBPROCESSDEFER	<p>Dubs each subtask of the caller as a new process when it issues its first z/OS UNIX service call. Dub processing is deferred until the next syscall is issued.</p> <p>This option must be issued from the job step task. If it is issued from any other task, the call is ignored.</p>	Yes	Yes	No
DUBJOBPERM	<p>Marks the process as permanent. The process is not terminated if F OMVS,SHUTDOWN is performed.</p> <p>To ensure that all processes in the address space are marked permanent, use this option from the job step task before tasks in the address space issues syscalls. When the system is shut down, any syscalls that are issued from the process will result in a failure.</p>	Yes	Yes	No

Table 29. Options and requirements for set\_dub\_default (continued)

Option	Function	Defers dubbing	Issue from job step task (JST)	Dub required
DUBABENDCALLS	<p>For processes that are permanent (dubbed with DUBJOBPERM), changes the default behavior from dub failure to abending the task when a syscall is issued while the system is being shut down.</p> <p>This option is only honored when DUBJOBPERM was also specified, either on the same call or another call. Like DUBJOBPERM, issue it from the job step task before any other processes are dubbed in the address space. Otherwise, the expected behavior might not occur for all tasks in the address space.</p>	Yes	Yes	No
DUBNOJSTUNDUB	<p>If the JST was blind-dubbed, then does not undub the JST (and end the process) when the last thread in the address space ends.</p> <p>Use this option to ensure that critical subsystem address spaces are not terminated during OMVS shutdown. Without this option, during shutdown or if an error causes a z/OS UNIX task to abnormally terminate, the entire job step is undubbed when the last dubbed task terminates.</p>	Yes	No	No
DUBUNIQUEACEE	<p>Indicates to the kernel that this address space does not share ACEEs between tasks for the life of the process. Address spaces that have tasks that redub can have the old USP associated with the ACEE deleted and a new USP obtained. Without this setting, a new USP is obtained, which then replaces the old USP. However, the old USP is not freed and the storage is lost until that address space ends.</p>	Yes	No	No
DUBFAILNOTREADY	<p>Any dubbing syscall that occurs during a non-kernel ready state will fail with the EMVSINITIAL return code or JRKernelReady reason code. For the syscalls that do not have a return code, an EC6 abend with reason code JNotUp is issued.</p> <p>After DUBFAILNOTREADY is issued from any task in the address space, all future dubs in that address space will fail instead of wait.</p>	Yes	No	No

### Usage notes for set\_dub\_default

1. DUBTHREAD is the default setting for a process that has not called this service.

2. This service can be called to override a previous call to the service.
3. When a task that was not already dubbed issues its first z/OS UNIX service call, its TCB tree is searched to determine the default dub setting to use. The search starts at the caller's mother task and continues up the TCB tree until an ancestor task is found that is already dubbed. If the search finds a dubbed task, the default dub setting from that task is used. If a dubbed task is not found, the task is dubbed as a new process. A dubbed task is a task that has one or more of the following attributes:
  - It has issued a z/OS UNIX service call.
  - It was created as a result of a fork service call.
  - It was created as a result of an exec or execmvs service call.
  - It was created as a result of an attach\_exec or attach\_execmvs service call.
  - It was created as a result of a pthread\_create service call.
4. DUBPROCESSDEFER is mutually exclusive with DUBPROCESS, DUBTASKACEE and DUBNOSIGNALS. Specifying DUBPROCESSDEFER with one or more of these options will cause the syscall to fail with -1/EINVAL/JRDUBSETTING.
5. If DUBNOSIGNALS is used in a POSIX(ON) environment, the behavior of the process is undefined.
6. DUBPROCESSDEFER should only be used from the job step task. It is not honored when issued from other tasks in the address space.
7. After the job step task (JST) issues a set\_dub\_default syscall with the DUBPROCESSDEFER option:
  - If the JST causes the address space to get dubbed, a new process is created and the JST is dubbed as the initial thread task (ITT) of the newly created process. The new process will have one thread, the JST.
  - If a task other than the JST causes the address space to be dubbed, a new process is created. The JST is dubbed as the ITT and the non-JST is dubbed as a thread in the newly created process. The new process will have two threads, the non-JST task and the JST that was dubbed as the ITT. The dubbing of the JST is known as *blind dubbing*. The JST must be dubbed if any other task in the address space is dubbed.
8. The DUBJOBPERM, DUBABENDCALLS, and DUBNOJSTUNDUB options should be used from the job step task prior to the call to any other z/OS UNIX callable service that could dub the address space, or they may not have their intended effect.
9. The DUBUNIQUEACEE option indicates whether an application is sharing ACEEs in a MultiProcess MultiUser (MpMu) address space. When specified, this option indicates that each process in the address space with a task level ACEE has a unique ACEE. That is, the ACEEs are not shared between tasks. When specified, it allows z/OS UNIX processing to do a clean-up of certain control blocks during the redubbing of a task in the address space.
 

This option is honored only when it is specified on a BPX1SDD call prior to an address space dub. When issued from any task after the address space has already been dubbed, this option is ignored.

## Characteristics and restrictions

1. When you set the DUBTASKACEE option, each task is dubbed as a separate process and uses the task-level ACEE that was set up by the user. In this environment, there are numerous restrictions on which other services can be used. This environment is supported primarily to allow a server to access zFS files and socket services. You cannot use z/OS UNIX security functions, such as setuid. Threads that are created with pthread\_create do not inherit the identity of the parent. Fork and spawn do not work correctly.
2. Users of the DUBJOBPERM and DUBABENDCALLS options must meet the following requirements:
  - The calling address space must be a system started task address space.
  - The caller must be running authorized (APF-authorized, system key 0-7, or supervisor state).

If these requirements are not met, the service will fail with return code EPERM.

**Examples**

For an example that uses this callable service, see [“BPX1SDD \(setdubdefault\) example”](#) on page 1284.

## setegid (BPX1SEG, BPX4SEG) – Set the effective group ID

---

**Function**

The setegid callable service sets the effective group ID (GID) of a process.

**Requirements****Operation****Authorization:****Dispatchable unit mode:****Cross memory mode:****AMODE (BPX1SEG):****AMODE (BPX4SEG):****ASC mode:****Interrupt status:****Locks:****Control parameters:****Environment**

Supervisor state or problem state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SEG, (Group_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SEG with the same parameters.

**Parameters****Group\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the group ID that the calling process wishes to assume.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setegid service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setegid service stores the return code. The setegid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The setegid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The Group_ID that was specified is incorrect.
<b>EPERM</b>	The process does not have the appropriate privileges (see <a href="#">“Authorization” on page 8</a> ) to set the group ID.
<b>EMVSSAF2ERR</b>	The SAF call IRRSEG00 incurred an error.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setegid service stores the reason code. The setegid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETEGID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	4	GID is not defined to RACF
<b>8</b>	8	User not authorized to change GID
<b>8</b>	12	Internal error during RACF processing
<b>8</b>	16	Unable to establish recovery

## Usage notes

1. If Group\_ID is equal to the real group ID or saved set group ID of the process, the effective group ID is set to Group\_ID.
2. If Group\_ID is not the same as the real group ID, and the calling process has the appropriate privileges (see [“Authorization” on page 8](#)), the effective group ID is set to Group\_ID.
3. The setegid service does not change any supplementary group IDs of the calling process.

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“getegid \(BPX1GEG, BPX4GEG\) — Get the effective group ID” on page 263](#)

## seteuid (BPX1SEU, BPX4SEU)

- [“getgid \(BPX1GID, BPX4GID\) — Get the real group ID ” on page 265](#)
- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID” on page 762](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)

### Characteristics and restrictions

- If the seteuid service is issued from multiple tasks within one address space, use synchronization to ensure that the seteuid services are not performed concurrently. The execution of seteuid service calls concurrently within one address space can yield unpredictable results.
- If the seteuid service is issued from an address space with multiple processes, the result of the service call affects all processes in the address space.

### Examples

See [“BPX1SEG \(setegid\) example” on page 1285](#).

## seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID

---

### Function

The seteuid callable service sets the effective user ID (UID) of a process.

### Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1SEU)	31-bit
AMODE (BPX4SEU)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1SEU, (User_ID,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4SEU with the same parameters.

### Parameters

#### User\_ID

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the user ID that the process is to assume.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the seteuid service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the seteuid service stores the return code. It returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\) in z/OS UNIX System Services Messages and Codes](#). One of the following values can be returned in the Return\_code parameter.

Return_code	Explanation
<b>EINVAL</b>	The User_ID that was specified is incorrect.
<b>EPERM</b>	The process does not have the appropriate privileges to set the user ID. Refer to <a href="#">“Authorization” on page 8</a> for information on appropriate privileges.
<b>EMVSSAF2ERR</b>	The SAF call IRRSEU00 incurred an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the seteuid service stores the reason code. The seteuid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#). The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETEUID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	UID is not defined to RACF
8	8	User not authorized to change UID
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

## Usage notes for seteuid

1. If a UNIX set-user-ID privileged program switches its UID to other than that of the set-user-ID program and it causes the real, effective and saved UID of the caller to be equal, then the set-user-ID privilege of the program is given up.
2. If User\_ID is the same as the real or saved set user ID of the process, or if the user has the appropriate privilege, the seteuid service sets the effective user ID to be the same as User\_ID. Refer to [“Authorization” on page 8](#) for information on appropriate privileges.
3. For information about changing MVS identities, and other restrictions, see the usage notes for [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#).
4. To attach the security environment of the caller of the setuid service to the security environment of the target UID (which then creates a nested ACEE for the target), use the \_BPXK\_DAEMON\_ATTACH environment variable. The new client can then access RACF delegated resources for which the daemon, but not necessarily the client, has access. (The delegated resources are designated by the APPDATA text of 'RACF-DELEGATED' in the RACF profile protecting the resource.) For more information about delegated resources, see [Defining delegated resources in z/OS Security Server RACF Security Administrator's Guide](#).
5. The seteuid service is allowed from a TSO environment. However, care should be taken when using this service since it can change the effective UID for the whole address space.

## Related services

- [“geteuid \(BPX1GEU, BPX4GEU\) — Get the effective user ID” on page 264](#)
- [“getuid \(BPX1GUI, BPX4GUI\) — Get the real user ID ” on page 329](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)

## Characteristics and restrictions

See the list of restrictions in [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#).

## Examples

See [“BPX1SEU \(seteuid\) example” on page 1286](#).

# setgid (BPX1SGI, BPX4SGI) — Set the group ID

---

## Function

The setgid callable service sets the real, effective, and saved set group IDs (GIDs) for the calling process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SGI):</b>	31-bit
<b>AMODE (BPX4SGI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked



**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SGI, (Group_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SGI with the same parameters.

**Parameters****Group\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the group ID that the calling process is to assume.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setgid service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setgid service stores the return code. The setgid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setgid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The Group_ID that was specified is incorrect.
<b>EPERM</b>	The process does not have the appropriate privileges to set the group ID. Refer to <a href="#">“Authorization” on page 8</a> for information about appropriate privileges.
<b>EMVSSAF2ERR</b>	The SAF call IRRSSG00 incurred an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setgid service stores the reason code. The setgid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#). The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETGID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	GID is not defined to RACF
8	8	User not authorized to change GID
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

**Usage notes**

1. If Group\_ID is equal to the real group ID or saved set group ID of the process, the effective group ID is set to Group\_ID.
2. If Group\_ID is not the same as the real group ID, and the calling process has the appropriate privileges, the real, saved set, and effective group IDs are set to Group\_ID. Refer to [“Authorization” on page 8](#) for information about appropriate privileges.
3. The setgid service does not change any supplementary group IDs of the calling process.

**Related services**

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“getegid \(BPX1GEG, BPX4GEG\) — Get the effective group ID” on page 263](#)
- [“getgid \(BPX1GID, BPX4GID\) — Get the real group ID ” on page 265](#)
- [“setegid \(BPX1SEG, BPX4SEG\) — Set the effective group ID ” on page 758](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)

**Characteristics and restrictions**

- The calling process must be privileged in order to change the real group ID—that is, to specify a group ID that is different from the process's real group ID. Refer to [“Authorization” on page 8](#) for information about appropriate privileges.
- If the setgid service is issued from multiple tasks within one address space, use synchronization to ensure that the setgid services are not performed concurrently. The execution of setgid services concurrently within one address space can yield unpredictable results.
- If the setgid service is issued from an address space with multiple processes, the result of the service call affects all processes in the address space.

**Examples**

See [“BPX1SGI \(setgid\) example” on page 1287](#).

## setgrent (BPX1SGE, BPX4SGE) – Reset the group database

---

### Function

The setgrent callable service resets the group database for subsequent searching by the getgrent service. The next getgrent service that is used after setgrent starts searching from the beginning of the group database.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SGE):</b>	31-bit
<b>AMODE (BPX4SGE):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1SGE, (Return_value)
```

AMODE 64 callers use BPX4SGE.

### Parameters

#### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setgrent service returns 0.

### Usage notes

The setgrent service is intended to be used to interrupt a sequential search of the group database from the calling task. The getgrent service performs the sequential search. When the setgrent service is called, it resets the search point for the current task in the group database to the beginning. The next getgrent service that is called from this task after this point starts searching the group database from the beginning.

### Related services

- [“getgrent \(BPX1GGE, BPX4GGE\) — Sequentially access the group database”](#) on page 266

## Characteristics and restrictions

None.

## Examples

See “BPX1SGE (setgrent) example” on page 1286.

# setgroups (BPX1SGR, BPX4SGR) – Set the supplementary group IDs list

---

## Function

The setgroups callable service replaces the existing supplementary group IDs (GIDs) list for the calling process with the list that is specified by the caller.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SGR):</b>	31-bit
<b>AMODE (BPX4SGR):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SGR,(SGid_list_count,
              SGid_list,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SGR with the same parameters. SGid\_list is a 64-bit pointer field.

## Parameters

### SGid\_list\_count

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that specifies the number of fullword entries in list that is pointed to by the SGid\_list parameter. The value must be in the range of zero to NGroups\_Max, inclusive.

Specifying 0 causes all existing supplementary group IDs for the calling process to be deleted. After the setgroups service completes, the calling process does not have any supplementary group IDs.

**SGid\_list**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains a pointer to an array of group IDs (GIDs). The setgroups service uses this list to establish the list of supplementary group IDs. The number of entries in the list is defined by the SGid\_list\_count parameter.

If the SGid\_list\_count specified is 0, the SGid\_list is ignored and does not need to contain a valid address.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setgroups service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setgroups service stores the return code. The setgroups service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setgroups service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	The SGid_list and SGid_list_count specify an array that is partially or completely outside the addressable storage range.
<b>EINVAL</b>	The SGid_list_count parameter was less than 0 or greater than NGroups_Max.
<b>EMVSSAF2ERR</b>	System authorization facility (SAF) had an error.
<b>EPERM</b>	The caller is not authorized; only authorized users are allowed to alter the supplementary group IDs list.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setgroups service stores the reason code. The setgroups service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

To determine the value of NGroups\_Max, see [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options”](#) on page 913.

## Related services

- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID”](#) on page 762
- [“getgroups \(BPX1GGR, BPX4GGR\) — Get a list of supplementary group IDs”](#) on page 274
- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options”](#) on page 913

## Characteristics and restrictions

- To set the supplementary group IDs, the requester must be a superuser. If a non-superuser caller requests the setgroups service, the service returns an EPERM Return\_code.
- To successfully complete the setgroups service, the caller's process must be the only process in the address space. If multiple processes are present (through attach\_exec or attach\_execMVS), the function does not complete successfully.

## Examples

See [“BPX1SGR \(setgroups\) example”](#) on page 1287.

# setitimer (BPX1STR, BPX4STR) — Set the value of the iterval timer

## Function

The setitimer callable service sets the timer value and optionally returns a pointer to a structure that contains the previous timer value. This function also generates a signal that is to be delivered when the interval timer expires.

## Requirements

Operation	Environment
<b>Authorization:</b>	Problem Program or Supervisor State, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1STR):</b>	31-bit
<b>AMODE (BPX4STR):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1STR (Interval_Type,
              Interval_Value_Adr,
              Old_Interval_Value_Adr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STR with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Interval\_Type

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a numeric value that identifies the interval timer (these values apply to both Interval\_Value\_Adr and Old\_Interval\_Value\_Adr). This parameter can have the following values:

#### ITIMER\_REAL

Real time (the default if VIRTUAL and PROF are not specified)

#### ITIMER\_VIRTUAL

Virtual time (CPU time minus system time)

#### ITIMER\_PROF

CPU time

#### ITIMER\_MICRO

The initial and reload times are in microseconds (the default if NANO is not specified)

#### ITIMER\_NANO

The initial and reload times are in nanoseconds

The ITIMER\_ constants are defined in the BPXYITIM macro.

### Interval\_Value\_Adr

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

A fullword (doubleword) field containing the address of a structure that is defined by the BPXYITIM macro. This structure contains the initial interval and reload values in seconds and either microseconds or nanoseconds.

### Old\_Interval\_Value\_Adr

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

A fullword (doubleword) field containing the address of a structure that is defined by the BPXYITIM macro. This structure contains the time remaining and reload values in seconds and either

microseconds or nanoseconds. This address may be zero if the current values are of no interest to the user.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setitimer service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setitimer service stores the return code. The setitimer service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setitimer service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• The value that was specified for Interval_Type is incorrect (JrStrIntervalTypeInvalid).</li> <li>• The value that was specified in the structure that is pointed to by Interval_Value_Adr is incorrect (JrNanoSecondsTooBig, JrMSecondsTooBig, JRNegativeValueInvalid).</li> </ul>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setitimer service stores the reason code. The setitimer service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The time between signals is maintained as a priority over the number of signals in an extended period of time. A delay in processing could result in a late signal; the signal generated by the reload value maintains the requested interval. As with other signals, multiples are lost. For example, for a 1 second timer, delays might result in 3597 signals per hour, with no two timers closer than 1 second. If the application requires exactly 3600 signals per hour, a reload value of zero should be used, and a new setitimer should be calculated and issued by the signal handler.
2. The duration between signals is always greater than, or equal to, the reload value that is specified.
3. Intervals vary, depending on when MVS gives the task its time slices.



4. The setting of the first two words of the Interval\_Value disables the timer, regardless of the reload value (third and fourth words).
5. Any setitimer() cancels the previous timer of the same type (that is, REAL, VIRTUAL, or PROF).
6. Real interval timers and alarms are treated as mutually exclusive. If an ITIMER\_REAL interval timer is issued while an alarm is set, the ITIMER\_REAL interval timer overlays the alarm, and vice versa.
7. The setitimer environment is propagated on the exec() and not propagated on fork().
8. Below are the interval timers and the corresponding signals that are to be generated when the timer expires:
  - ITIMER\_REAL, which decrements in real time. A SIGALRM signal is delivered when this timer expires.
  - ITIMER\_VIRTUAL, which decrements in task virtual time. It runs only when the task is executing outside the kernel. A SIGVTALRM signal is delivered when it expires. Task virtual is a best estimate, and loses significance when it is run in a multiprocess environment.
  - ITIMER\_PROF, which decrements in task time. It runs when the task is running on behalf of the process. A SIGVPROF signal is delivered when it expires.
9. The setitimer(), alarm(), and sleep() services use the MVS STIMER macro. If the task invokes the STIMER macro and a combination of these services, the limit of concurrent STIMER SET requests for a task can be exceeded, which results in an abnormal end.
10. ITIMER\_REAL interval timers are supported in both multiprocess and multithreaded environments.
11. You can issue Setitimer() for ITIMER\_PROF or ITIMER\_VIRTUAL in a multithreaded or multiprocess environment. However, for ITIMER\_VIRTUAL in a multithread environment, the results may be unpredictable.
12. If two interval timers of the same type expire before a signal is delivered, only one signal is generated.
13. The reload time is set before the signal interface routine is given control.

### MVS-related information

- [“getitimer \(BPX1GTR, BPX4GTR\) — Get the value of the interval timer” on page 287](#)
- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)

### Characteristics and restrictions

None (other than those indicated in the usage notes).

### Examples

See [“BPX1STR \(setitimer\) example” on page 1298](#).

## setns (BPX1SNS, BPX4SNS) — Reassociate thread with a namespace

---

### Function

The setns service moves a process into an existing namespace. For namespaces other than PID namespaces, the current process is moved into the specified namespace as a result of the setns service. For PID namespaces, subsequent child processes that were created by the current process after the setns request are associated with the specified namespace.

For the macro, see [“BPXYCLNP— Map clone syscall parameters” on page 1062](#).

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SNS)</b>	31-bit
<b>AMODE (BPX4SNS)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and be in the primary address space.

## Format

```
CALL BPX1SNS, (Ns_File_Descriptor,
               Nstype,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SNS with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Ns\_File\_Descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that contains the file descriptor referring to the /proc[pid]/ns magic link (/pid, /mnt, /net, /ipc, /uts) for the specific namespace type.

### Nstype

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword field that specifies the type of namespace to which the calling process can be reassociated.

When the Ns\_File\_Descriptor refers to a /proc/[pid]/ns link, only a single flag that matches the namespace type referred to by the file descriptor or a 0 flag can be specified.

The flags are mapped by the CLONE\_FLAGS structure in the BPXYCLNP macro. For more information, see [“BPXYCLNP— Map clone syscall parameters” on page 1062](#). The following flags are supported by the setns service:

Flag	Description
0	The process is reassociated into the namespace type that is associated with the input file descriptor.
CLONE_NEWIPC	Reassociate the process into the IPC namespace associated with the input file descriptor.  If specified, <code>Ns_File_Descriptor</code> must refer to either an IPC link in the <code>/proc/[pid]/ns</code> directory.
CLONE_NEWNS	Reassociate the process into the mount namespace associated with the input file descriptor.  If specified, <code>Ns_File_Descriptor</code> must refer to either a mount link in the <code>/proc/[pid]/ns</code> directory.
CLONE_NEWPID	Reassociate the process into the IPC namespace associated with the input file descriptor.  If specified, <code>Ns_File_Descriptor</code> must refer to either a PID link in the <code>/proc/[pid]/ns</code> directory.
CLONE_NEWUTS	Reassociate the process into the UTS namespace associated with the input file descriptor.  If specified, <code>Ns_File_Descriptor</code> must refer to either a UTS link in the <code>/proc/[pid]/ns</code> directory.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the `unshare` service returns 0 if the request is successful or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the **unshare** service stores the return code. The **unshare** service returns `Return_code` only if `Process_ID` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. A list of possible return codes follows.

Return_code	Explanation
<b>EBADF</b>	<code>Ns_File_Descriptor</code> is not valid.
<b>EINVAL</b>	One of the input parameters was not valid.  The following reason code can accompany the return code: <code>JRUnsupportedFlag</code> .
<b>ENOMEM</b>	The process requires more space than is available.
<b>ESRCH</b>	The process referred to by the file descriptor no longer exists.

Return_code	Explanation
<b>EPERM</b>	<p>The calling process does not have appropriate privileges.</p> <ul style="list-style-type: none"> <li>The user is not a superuser and is not permitted to the CONTAINERS resource in the UNIXPRIV class.</li> </ul> <p>The following reason code can accompany the return code: JrNotAuthNameS.</p>
<b>EAGAIN</b>	<p>The resources required to let another process be created are not available now or you have reached the maximum number of processes you are allowed to run.</p> <p>The following reason code can accompany the return code: JrNoSecurityProduct.</p>
<b>EMVSSAF2ERR</b>	<p>An error occurred in the security product.</p> <p>The following reason code can accompany the return code: JrSAFInternal.</p>

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the unshare service returns 0 if the request is successful, or -1 if it is not successful.

**Usage notes**

1. The setns service supports the following flags for the nstype parameter: CLONE\_NEWNS, CLONE\_NEWPID, CLONE\_NEWIPC, and CLONE\_NEWUTS. Unsupported flags will result in the operation failing with an EINVAL error.
2. The caller must be authorized by being a superuser or having at least READ access to the CONTAINERS resource in the UNIXPRIV class.
3. For setns requests that specify CLONE\_NEWNS, CLONE\_NEWIPC, or CLONE\_NEWUTS, the current process is moved into the namespace associated with the specified file descriptor. For setns requests that specify CLONE\_NEWPID, the current process does not change its PID namespace. Instead, subsequent children of the calling process are associated with the PID namespace represented by the Ns\_File\_Descriptor.

**Related services**

- [“clone \(BPX1CLN, BPX4CLN\) — Create a child process” on page 114](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)
- [“unshare \(BPX1UNS, BPX4UNS\) — Run program in a new namespace” on page 977](#)

**Examples**

For examples that use the setns callable service, see [“BPX1SNS \(setns\) example” on page 1291](#) and [“BPX4SNS \(setns\) example” on page 1384](#).

# setpeer (BPX1SPR, BPX4SPR) – Preset the peer address associated with a socket

## Function

The setpeer callable service presets the peer address that is associated with a socket.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SPR):</b>	31-bit task or SRB mode
<b>AMODE (BPX4SPR):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SPR, (Socket_descriptor,
               Sockaddr_length,
               Sockaddr,
               Option,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SPR with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the setpeer is to be done.

### Sockaddr\_length

Supplied and returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the length of Sockaddr.

**Sockaddr**

Supplied and returned parameter

**Type:**

Structure

**Length:**

Length specified by Sockaddr\_length

The name of a socket address structure that contains the peer address. For more information about this field, see [BPXYSOCK](#).

**Option**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that indicates the conditions of the setpeer request. For more information about this field, see [BPXYSOCK](#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setpeer service returns one of the following:

- 0 if the request is successful.
- -1 if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Character set:**

N/A

**Length:**

Fullword

The name of a fullword in which the setpeer service stores the return code. The setpeer service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setpeer service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setpeer service stores the reason code. The setpeer service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The setpeer service is not supported by AF\_UNIX, AF\_INET, or AF\_INET6.
2. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1SPR \(setpeer\) example” on page 1294](#).

## setpgid (BPX1SPG, BPX4SPG) – Set a process group ID for job control

---

**Function**

The setpgid callable service places a process in a process group. You identify the group by specifying a process group ID. You can assign a process to a different group, or you can start a new group with that process as its leader.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SPG):</b>	31-bit
<b>AMODE (BPX4SPG):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SPG, (Process_ID,
               Process_group_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SPG with the same parameters.

## Parameters

### Process\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the ID of the process that is to be placed in the process group. If the ID is specified as 0, the system uses the process ID of the calling process.

### Process\_group\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the ID of the process group where Process\_ID is assigned. If the ID is specified as 0, the system uses the process group ID that is indicated by the Process\_ID parameter.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setpgid service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setpgid service stores the return code. The setpgid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setpgid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The value of Process_ID matches the process ID of a child of the calling process, but the child has successfully invoked one of the exec functions. Access to the target process was denied. The following reason code can accompany the return code: JRExecAfterFork.
<b>EINVAL</b>	The Process_group_ID parameter is less than zero or has some other unsupported value. The following reason codes can accompany the return code: JRNoSuchPid and JRPgidDifferentSession.



Return_code	Explanation
<b>EPERM</b>	The calling process cannot change the process group ID of the specified process. The following reason codes can accompany the return code: JRPidEQSessLeader, JRPidDifferentSession, and JRPgidDifferentSession.
<b>ESRCH</b>	The specified Process_ID is not that of the calling process or any of its children. The following reason codes can accompany the return code: JRNotDescendant and JRNoSuchPid.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setpgid service stores the reason code. The setpgid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The process group ID that is to be assigned to the group must be within the calling process's session.
2. The subject process (the process identified by the Process\_ID parameter) must be a child of the process that issues the service and must be in the same session, but it cannot be the session leader. It can be the caller.

**Related services**

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“getpgrp \(BPX1GPG, BPX4GPG\) — Get the process group ID” on page 297](#)
- [“setsid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ” on page 793](#)
- [“tcsetpgrp \(BPX1TSP, BPX4TSP\) — Set the foreground process group ID ” on page 944](#)

**Characteristics and restrictions**

See the conditions described under Return\_code.

**Examples**

See [“BPX1SPG \(setpgid\) example” on page 1293](#).

## setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process

---

**Function**

The setpriority callable service sets the scheduling priority of a process, process group, or user.

**Requirements****Operation****Environment****Authorization:**

Supervisor or problem state, any PSW key

## setpriority (BPX1SPY, BPX4SPY)

Operation	Environment
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SPY):</b>	31-bit
<b>AMODE (BPX4SPY):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1SPY,(Which,
               Who,
               Priority,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SPY with the same parameters.

### Parameters

#### Which

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains a value that indicates how the Who parameter is to be interpreted. This parameter can have one of the following values:

#### **PRIO\_PROCESS**

The Who parameter is to be interpreted as a process ID.

#### **PRIO\_PGRP**

The Who parameter is to be interpreted as a process group ID.

#### **PRIO\_USER**

The Who parameter is to be interpreted as a user ID.

The PRIO\_ constants are defined in the BPXYCONS macro. See [“BPXYCONS — Constants used by services” on page 1062](#).

#### Who

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains a value that indicates the exact process ID, process group ID, or user ID whose priority is to be set. The Which parameter indicates how this parameter is to be interpreted. A value of zero for this parameter specifies the current process, process group, or user ID.

**Priority**

Supplied parameter

**Type:**

Signed Integer

**Length:**

Fullword

The name of a fullword that contains a value that indicates the priority value to which the specific process or group of processes is to be set. This value can be an integer in the range of -20 to 19.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setpriority service returns a value of zero if successful and -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setpriority service stores the return code. The setpriority service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setpriority service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The priority is being changed to a lower value, and the current process does not have the appropriate privilege (see <a href="#">“Authorization” on page 8</a> ) to do so.
<b>EINVAL</b>	The value of the Which parameter was not recognized; or the value of the Who parameter is not a valid process ID, process group ID or user ID.
<b>EMVSAF2ERR</b>	A security product internal error occurred. Consult the Reason_code parameter for the exact reason for the error.
<b>ENOSYS</b>	The system does not support this function. Your installation chose not to enable this function.
<b>EPERM</b>	A process was located, but neither the real nor the effective user ID of the calling process matches the effective user ID of the process whose priority is being changed.
<b>ESRCH</b>	No process could be located using the Which and Who parameter values that were specified.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setpriority service stores the reason code. The setpriority service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the supplied Who and Which values specify more than one process, each of the specified processes has its priority value set to the supplied value. If at least one of the specified processes has its priority value successfully changed, the setpriority service returns successfully.
2. The priority value of a process is an integer that can be in the range of -20 to 19. If the supplied priority value is outside this range, the process's priority is set to the corresponding limit value. The default priority value for all processes is 0.
3. An increase in a process's priority value results in a lower CPU priority for the process. A decrease in a process's priority value results in a higher CPU priority for the process.
4. If the supplied priority value would result in a lower priority value for the specified process or processes, the caller must have appropriate privileges. Refer to [“Authorization” on page 8](#) for information about appropriate privileges.
5. The setting of a process's priority value has a corresponding effect on its nice value because they both represent the process's relative CPU priority. For example, using the setpriority service to set the priority value of a process to its maximum value (19) has the effect of increasing its nice value to its maximum value (2\*NICE\_ZERO)-1, and this is reflected on the nice, getpriority, and setpriority services. The NICE\_ZERO constant is defined in BPXYCONS.
6. If the ENOSYS return code is received, your installation does not support this service. Contact your system administrator if you require activation of this service.
7. If the supplied Who and Which values specify a process in a multiple process address space, each of the processes in the address space have their priority values set to the supplied value.
8. To do the initial system setup for using this service, see [Enabling nice\(\)](#), [setpriority\(\)](#), and [chpriority\(\)](#) support in *z/OS UNIX System Services Planning*.

**Related services**

- [“nice \(BPX1NIC, BPX4NIC\) — Change the nice value of a process” on page 497](#)
- [“getpriority \(BPX1GPY, BPX4GPY\) — Get the scheduling priority of a process ” on page 300](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1SPY \(setpriority\) example” on page 1295](#).

## **setpwent (BPX1SPE, BPX4SPE) — Reset the user database**

---

**Function**

The setpwent callable service resets the user database for subsequent searching by the getpwent service. The next getpwent service that is used after setpwent starts searching from the beginning of the user database.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SPE):</b>	31-bit
<b>AMODE (BPX4SPE):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SPE, (Return_value)
```

AMODE 64 callers use BPX4SPE.

## Parameters

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setpwent service returns 0.

## Usage notes

The setpwent service is intended to be used to interrupt a sequential search of the user database from the calling task. The getpwent service performs the sequential search. When the setpwent service is called, it resets the search point for the current task in the user database to the beginning. The next getpwent service that is called from this task after this point starts searching the user database from the beginning.

## Related services

- [“getpwent \(BPX1GPE, BPX4GPE\) — Sequentially access the user database” on page 302](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1SPE \(setpwent\) example” on page 1292](#).

## setregid (BPX1SRG, BPX4SRG) – Set the real and effective GIDs

### Function

The setregid callable service sets the real or effective GIDs for the calling process to the values that are specified by the input real and effective GID values. If a specified value is equal to -1, the corresponding real or effective GID of the calling process is left unchanged.

### Requirements

#### Operation

#### Authorization:

#### Dispatchable unit mode:

#### Cross memory mode:

#### AMODE (BPX1SRG):

#### AMODE (BPX4SRG):

#### ASC mode:

#### Interrupt status:

#### Locks:

#### Control parameters:

#### Environment

Supervisor or problem state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1SRG, (RGID,
               EGID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SRG with the same parameters.

### Parameters

#### RGID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the real GID to be set for the calling process. If RGID is -1, the real GID for the calling process is left unchanged.

#### EGID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the effective GID to be set for the calling process. If EGID is -1, the effective GID for the calling process is left unchanged.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setregid service returns -1 if it is not successful. If it is successful, the setregid service returns 0.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setregid service stores the return code. The setregid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The setregid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value of RGID or EGID is not valid.
<b>EPERM</b>	The process does not have appropriate privileges to set the real GID or the effective GID. Refer to <a href="#">“Authorization” on page 8</a> for information on appropriate privileges.
<b>EMVSSAF2ERR</b>	The SAF call IRRSSG00 incurred an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setregid service stores the reason code. The setregid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETGID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

For a more detailed description of the RACF CKPRIV service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	12	Internal error during RACF processing

## Usage notes

1. A process with appropriate privileges (see “Authorization” on page 8) can set the real and effective GID to any valid GID value. An unprivileged process can only set the effective GID if the EGID argument is equal to the real, effective, or saved GID of the process. An unprivileged process can only set the real GID if the RGID argument is equal to the real, effective, or saved GID of the process.
2. The setregid does not change any supplementary GIDs of the calling process.

## Related services

- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 163
- “getegid (BPX1GEG, BPX4GEG) — Get the effective group ID” on page 263
- “setgid (BPX1SGI, BPX4SGI) — Set the group ID” on page 762
- “getuid (BPX1GUI, BPX4GUI) — Get the real user ID ” on page 329
- “setuid (BPX1SUI, BPX4SUI) — Set user IDs” on page 800
- “setreuid (BPX1SRU, BPX4SRU) —Set the real and effective UIDs” on page 786

## Characteristics and restrictions

- If the setregid service is issued from multiple tasks within one address space, use synchronization to ensure that the setregid services are not performed concurrently. The execution of setregid requests concurrently within one address space can yield unpredictable results.
- If the setregid service is issued from an address space with multiple processes, the result of the service call affects all processes in the address space.

## Examples

See “BPX1SRG (setregid) example” on page 1295.

# **setreuid (BPX1SRU, BPX4SRU) —Set the real and effective UIDs**

---

## Function

The setreuid callable service sets the real and effective UIDs for the calling process to the values that are specified by the input real and effective UID values. If a specified value is equal to -1, the corresponding real or effective UID of the calling process is left unchanged.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SRU):</b>	31-bit
<b>AMODE (BPX4SRU):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.



## Format

The syntax format is as follows:

```
CALL BPX1SRU, (RUID,
               EUID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SRU with the same parameters.

## Parameters

### RUID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the real UID to be set in the calling process. If RUID is -1, the real UID for the calling process is left unchanged.

### EUID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the effective UID to be set in the calling process. If EUID is -1, the effective UID for the calling process is left unchanged.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setreuid service returns -1 if it is not successful. If it is successful, the setreuid service returns 0.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setreuid service stores the return code. The setreuid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setreuid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EINVAL	The value of RUID or EUID is not valid.

Return_code	Explanation
<b>EPERM</b>	The process does not have appropriate privileges to set the real UID or the effective UID. Refer to <a href="#">“Authorization” on page 8</a> for information on appropriate privileges.
<b>EMVSSAF2ERR</b>	The SAF call IRRSSU00 incurred an error.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setreuid service stores the reason code. The setreuid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see Reason codes in *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETUID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	12	Internal error during RACF processing
<b>8</b>	16	Unable to establish recovery

For a more detailed description of the RACF CKPRIV service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
<b>8</b>	4	User is not privileged
<b>8</b>	12	Internal error during RACF processing

**Usage notes for setreuid**

1. If a UNIX set-user-ID privileged program switches its UID to other than that of the set-user-ID program and it causes the real, effective and saved UID of the caller to be equal, then the set-user-ID privilege of the program is given up.
2. A process with appropriate privileges (see [“Authorization” on page 8](#)) can set the real and effective UID to any valid UID value. An unprivileged process can set the effective UID only if the EUID argument is equal to the real, effective, or saved UID of the process. An unprivileged process can set the real UID only if the RUID argument is equal to the real, effective, or saved UID of the process.
3. The setreuid service is allowed in a TSO address space so long as the caller does not attempt to change the MVS identity. MVS identity changes are triggered by changing the effective UID. The real UID can always be changed if the invoker has appropriate privileges.
4. For information about changing MVS identities, and other restrictions, see the UsageNotes for [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#).
5. To attach the security environment of the caller of the setuid service to the security environment of the target UID (which then creates a nested ACEE for the target), use the `_BPXK_DAEMON_ATTACH` environment variable. The new client can then access RACF delegated resources for which the daemon, but not necessarily the client, has access. (The delegated resources are designated by the APPDATA text of 'RACF-DELEGATED' in the RACF profile protecting the resource.) For more information about delegated resources, see [Defining delegated resources in z/OS Security Server RACF Security Administrator's Guide](#).

6. The setreuid service is allowed from a TSO environment. However, care should be taken when using this service since it can change the effective UID for the whole address space.

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“geteuid \(BPX1GEU, BPX4GEU\) — Get the effective user ID” on page 264](#)
- [“getuid \(BPX1GUI, BPX4GUI\) — Get the real user ID ” on page 329](#)
- [“seteuid \(BPX1SEU, BPX4SEU\) — Set the effective user ID” on page 760](#)
- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID” on page 762](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)
- [“setregid \(BPX1SRG, BPX4SRG\) — Set the real and effective GIDs ” on page 784](#)

## Characteristics and restrictions

See the list of restrictions in [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#).

See also [“Usage notes for setreuid” on page 788](#).

## Examples

See [“BPX1SRU \(setreuid\) example” on page 1296](#).

# setrlimit (BPX1SRL, BPX4SRL) — Set resource limits

---

## Function

The setrlimit callable service sets resource limits for the calling process. A resource limit is a pair of values; one specifies the current (soft) limit and the other the maximum (hard) limit.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SRL)</b>	31-bit
<b>AMODE (BPX4SRL)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SRL,(Resource,
               Rlimit,
               Return_value,
```

Return\_code,  
Reason\_code)

AMODE 64 callers use BPX4SRL with the same parameters.

## Parameters

### Resource

Supplied parameter.

### Type

Integer

### Length

Fullword

The name of a fullword that contains a value that indicates the resource for which to set the hard and soft limits. This parameter can specify one of the resources in listed in [Table 30 on page 790](#):

<i>Table 30. Resources that can be limited by setrlimit</i>		
Resource	Description	Action when soft limit is exceeded
<b>RLIMIT_MEMLIMIT</b>	Maximum amount of usable storage above the bar (in 1-MB segments) that can be allocated at one time.	Any attempt to use more above the bar storage than is specified by the limit will fail.
<b>RLIMIT_CORE</b>	Maximum core file size (in bytes) created by a process. A value of 0 prevents core file creation.	Core file writing terminates at this size.
<b>RLIMIT_CPU</b>	Maximum amount of CPU time (in seconds) used by a process.	SIGXCPU is sent to the process, and the process is granted a small extension to allow for signal generation and delivery. If the extension is used up, the process is terminated with a SIGKILL.
<b>RLIMIT_FSIZE</b>	Maximum file size (in bytes) created by a process. A value of 0 prevents the creation of new files and the expansion of existing files.	SIGXFSZ is sent to the process. If the process is blocking, catching, or ignoring SIGXFSZ, continued attempts to increase the size of a file beyond the limit fail with a return code of EFBIG.
<b>RLIMIT_NOFILE</b>	Maximum number of open file descriptors for a process. This number is one greater than the maximum value that may be assigned to a newly created descriptor.	Functions that create new file descriptors after the limit is reached fail with a return code of EMFILE.
<b>RLIMIT_AS</b>	Maximum address space size (in bytes) for a process.	The mmap and shmat callable services fail with a return code of ENOMEM. User getmain and storage obtain requests fail (for example, runtime library stack and heap expansion fails).

The RLIMIT\_ constants are defined in the BPXYCONS macro. See [“BPXYCONS — Constants used by services” on page 1062](#).

### Rlimit

Supplied parameter.

**Type**

Structure

**Length**

The length of the rlimit structure

The name of an Rlimit structure that contains the values for the hard (maximum) and soft (current) limits for the resource that is identified by the resource parameter. Macro BPXYRLIM defines the Rlimit structure. (See [“BPXYRLIM — Map the rlimit, rusage, and timeval structures”](#) on page 1142.) Each limit value contains two fullwords. For all resources except RLIMIT\_FSIZE and RLIMIT\_MEMLIMIT, the upper fullword for each limit value is ignored.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the setrlimit service returns a value of zero if it is successful, and -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the setrlimit service stores the return code. The setrlimit service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The setrlimit service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	<p>The operation is not permitted for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• The resource that was specified is not valid.</li> <li>• The soft limit that is to be set exceeds the hard limit to set.</li> <li>• The soft limit that is to be set is below the current usage.</li> <li>• The hard limit that is to be set exceeds a system-defined limit.</li> <li>• One of the file size limits that was specified is a negative value.</li> </ul> <p>The following reason codes can accompany the return code: JrInvalidResource, JrSoftExceedsHard, JrSoftBelowUsage, JrFdOpenAboveLimit, JrOpenFileLimitMax, or JrNegFileSizeLimit.</p>
<b>EMVSSAF2ERR</b>	A Security product internal error has occurred. Consult the Reason_code parameter for the exact reason for the error.
<b>EPERM</b>	An attempt was made to raise a hard (maximum) limit, but the calling process did not have superuser authority. The following reason code can accompany the return code: JrRaiseHardLimit.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the setrlimit service stores the reason code. The setrlimit service returns Reason\_code only if Return\_value is -1. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The soft limit can be modified to any value that is less than or equal to the hard limit. For the RLIMIT\_CPU, RLIMIT\_NOFILE, and RLIMIT\_AS resources, if setrlimit is called with a soft limit that is lower than the current usage, setrlimit fails with an EINVAL return code.  
  
An exception to this rule occurs when the process is running in an address space that contains multiple processes. When you change the RLIMIT\_CPU, you can set a new soft limit that is greater than the time limit of the current process, yet greater than the time consumed by the address space. This allows the setrlimit call to succeed, and a SIGXCPU signal is generated. The alternative is not to run multiple processes in the same address space.
2. The hard limit may be lowered to any value that is greater than or equal to the soft limit.
3. The hard limit can only be raised by a process that has superuser authority.
4. Both the soft limit and the hard limit can be changed with a single call to the setrlimit service.
5. If the setrlimit service is called with a soft limit that is greater than the hard limit, setrlimit returns an EINVAL return code.
6. The resource limit values are propagated across the exec, fork, and spawn services. An exception exists for the exec and spawn for services. If a daemon process invokes the exec service and it has previously invoked setuid, or invokes the spawn service specifying that a user ID change is to occur, then the limit values are set based on the values that are found in the OMVS segment of the target user ID. If the target user's OMVS segment does not specify limit values, then the limit values are set based on the values that are found in the BPXPRMxx parmlib member.
7. For processes in a multiprocess address space, the RLIMIT\_CPU and RLIMIT\_AS limits are shared with all the processes within the address space. For RLIMIT\_CPU, when the soft limit is exceeded, action is taken on the first process within the address space. If the action is termination, all the processes within the address space are terminated.
8. In addition to the RLIMIT\_CORE limit values, CORE dump defaults are set by SYSMDUMP defaults. See [IEADMR00 \(ABDUMP written to a SYSMDUMP data set\)](#) in *z/OS MVS Initialization and Tuning Reference* information on setting up SYSMDUMP defaults via the IEADMR00 parmlib member.
9. Core dumps are taken in 4160-byte increments. Therefore, RLIMIT\_CORE values affect the size of core dumps in 4160-byte increments. For example, if the RLIMIT\_CORE soft limit value is 0, no core dumps are taken. If the RLIMIT\_CORE soft limit value is 8000, the maximum size of a core dump is 8000 \* 4160 bytes.
10. Limits may have an infinite value of RLIM\_INFINITY. MEMLIMIT cannot exceed 16383G, which is defined RLIM\_MEGINFINITY.
11. The hard limit for RLIMIT\_NOFILE cannot exceed the system-defined limit of 524287. A value of RLIM\_INFINITY for RLIMIT\_NOFILE indicates that the current system maximum value should be set.
12. The soft limit for RLIMIT\_NOFILE must be set higher than the value of the highest open file descriptor. Attempting to lower the soft limit to a value that is less than or equal to the highest open file descriptor results in an EINVAL return code.
13. Setting a limit of 0 for RLIMIT\_FSIZE prevents the creation of new files and the expansion of existing files.
14. When RLIM\_INFINITY (X'7FFFFFFF') is passed on a setrlimit request, no limit is enforced by setrlimit. As a result, the maximum allowable limit is set, regardless of the resource. The new service RLIM\_MEMLIMIT treats RLIM\_INFINITY as a request for 21474836471 1- megabyte pages.
15. When the MEMLIMIT is set by z/OS UNIX, the highest value that is supported is 16383 petabytes, or X'FFFC000000000000'.

16. Processes can use this service to control CPU resource consumption. For more information, see [BPXPRMxx \(z/OS UNIX System Services parameters\)](#) in *z/OS MVS Initialization and Tuning Reference*.

## Related services

- [“getrlimit \(BPX1GRL, BPX4GRL\) — Get resource limits ” on page 313](#)
- [“getrusage \(BPX1GRU, BPX4GRU\) — Get resource usage ” on page 315](#)
- [“prlimit \(BPX1PRL, BPX4PRL\) — Get and set resource limits for specified processes ” on page 574](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1SRL \(setrlimit\) example” on page 1295](#).

# setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID

---

## Function

The setsid callable service creates a new session, with the calling process as its session leader. The caller becomes the group leader of a new process group.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SSI):</b>	31-bit
<b>AMODE (BPX4SSI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SSI, (Session_ID,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SSI with the same parameters.

## Parameters

### Session\_ID

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword to which, if successful, the setsid service returns the session or process group ID of the new group. The new session or group process ID is the same as the process ID of the caller.

If not successful in creating a new session, the setsid service returns -1 as the Session\_ID value.

### Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setsid service stores the return code. The setsid service returns Return\_code only if Session\_ID is -1. SFor a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The setsid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EPERM</b>	The caller is already a process group leader, or the caller's process ID matches the process group ID of some other process. The following reason code can accompany the return code: JRCallerIsPgLeader.

### Reason\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setsid service stores the reason code. The setsid service returns Reason\_code only if Session\_ID is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

The calling process does not have a controlling terminal.

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program”](#) on page 163
- [“\\_exit \(BPX1EXI, BPX4EXI\) — End a process and bypass the cleanup ”](#) on page 180
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ”](#) on page 227
- [“getpid \(BPX1GPI, BPX4GPI\) — Get the process ID”](#) on page 298
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ”](#) on page 360
- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ”](#) on page 777
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ”](#) on page 834



## Characteristics and restrictions

The calling process must not already be a process group leader.

### Examples

See “BPX1SSI (setsid) example” on page 1296.

## setsockopt or getsockopt (BPX1OPT, BPX4OPT) – Get or set options associated with a socket

---

See “getsockopt or setsockopt (BPX1OPT, BPX4OPT) – Get or set options associated with a socket” on page 322.

## set\_thread\_limits (BPX1STL, BPX4STL) – Change task or thread limits for pthread\_created threads

---

### Function

The set\_thread\_limits callable service changes the calling process's limits for pthread\_created threads. These limits are the maximum number of MVS tasks used for pthread\_created threads, and the maximum number of pthread\_created threads. The thread limit includes running, queued, and undetached exited threads.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1STL):</b>	31-bit
<b>AMODE (BPX4STL):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1STL, (Action,
               MaxThreadTasks,
               MaxThreads,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4STL with the same parameters.

## Parameters

### Action

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a numeric value that identifies the process's pthread\_created thread limits that are to be set. The following constants, which are defined in BPXYCONS, define the actions that are to be taken (see [“BPXYCONS — Constants used by services”](#) on page 1062):

#### **STL\_MAX\_TASKS**

Replace the MaxThreadTasks limit for the caller's process with the value that is specified in MaxThreadTasks only.

#### **STL\_MAX\_THREADS**

Replace the MaxThreads limit for pthread\_created threads in the caller's process with the fullword value that is specified in MaxThreads only.

#### **STL\_SET\_BOTH**

Replace both the MaxThreadTasks and MaxThreads limits for the caller's process with the fullword values that are specified in MaxThreadTasks and MaxThreads, respectively.

### MaxThreadTasks

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

When the Action that is specified is STL\_MAX\_TASKS or STL\_SET\_BOTH, this is the name of a fullword that contains the new MaxThreadTasks value for the caller's process.

### MaxThreads

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

When the Action that is specified is STL\_MAX\_THREADS or STL\_SET\_BOTH, this is the name of a fullword that contains the new MaxThreads value for the caller's process.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the set\_thread\_limits service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the set\_thread\_limits service stores the return code. The set\_thread\_limits service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The set\_thread\_limits service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EINVAL**

The value that was specified for Action, MaxThreadTasks, or MaxThreads is incorrect. The following reason codes can accompany the return code: JRSTLActionInvalid, JRSTLTasksInvalid or JRSTLThreadsInvalid.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the set\_thread\_limits service stores the reason code. The set\_thread\_limits service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the set\_thread\_limits service returns with an unsuccessful return value (-1), the original MaxThreadTasks and MaxThreads values for the caller's process remain unchanged.
2. If any caller, authorized or nonauthorized, attempts to set a limit outside the allowable ranges (see [Table 32 on page 797](#)), the set\_thread\_limits service returns with a return code of EINVAL and a reason code of JRSTLTasksInvalid or JRSTLThreadsInvalid. Parmlib represents the values that are specified at z/OS UNIX startup by the BPXPRMxx parmlib member.
  - For MaxThreadTasks, the limits are shown in [Table 31 on page 797](#).

<i>Table 31. Allowable thread limits for MaxThreadTasks</i>		
Type of task	Minimum	Maximum
Authorized	1	32768
Unauthorized	1	Parmlib

- For MaxThreads, the limits are shown in [Table 32 on page 797](#). Parmlib represents the values that are specified at z/OS UNIX startup by the BPXPRMxx parmlib member.

<i>Table 32. Allowable thread limits for MaxThread</i>		
Type of tasks	Minimum	Maximum
Authorized	0	100000
Unauthorized	0	Parmlib

3. To determine the allowable ranges for pthread\_created thread limits for nonauthorized callers, see [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options” on page 913](#).
4. For information about setting initial thread limits and performance considerations, see [MAXTHREADS](#) in *z/OS UNIX System Services Planning*.

## set\_timer\_event (BPX1STE, BPX4STE)

5. If the MaxThreadTasks limit is decreased below the number of tasks that are currently in use, pthread\_exit\_and\_get requests fail until the number of tasks in use is less than or equal to the new limit.
6. Setting the MaxThreads limit to zero inhibits the creation of pthread\_created threads.
7. Setting MaxThreads to be less than or equal to MaxThreadTasks prevents the queueing of pthread\_create requests, and limits the number of MVS tasks that are attached for pthread\_created threads to the MaxThreads value.
8. If the MaxThreadTasks limit of a process is set below the number of MVS tasks that are already in use for pthread\_created threads, the reduction of MVS tasks is completed as running threads terminate. The reduction of tasks is not synchronously carried out when the set\_thread\_limits service is invoked.
9. For POSIX compliance, the MaxThreads limit for a process must be 64 or greater.

### Related services

- [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread” on page 581](#)
- [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ” on page 913](#)
- [“pthread\\_exit\\_and\\_get \(BPX1PTX, BPX4PTX\) — Exit and get a new thread ” on page 588](#)

### Characteristics and restrictions

None.

### Examples

See [“BPX1STL \(set\\_thread\\_limits\) example” on page 1298](#).

## set\_timer\_event (BPX1STE, BPX4STE) — Set DIE-mode timer event

### Function

The set\_timer\_event callable service sets a DIE-mode timer event that posts an ECB when it expires. The ECB is located in the BPXYTHLI data area.

### Requirements

Operation	Environment
<b>Authorization:</b>	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1STE):</b>	31-bit
<b>AMODE (BPX4STE):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1STE, (Seconds,
               Nanoseconds,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4STE with the same parameters.

## Parameters

### Seconds

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains an unsigned integer that represents the maximum number of seconds that the calling program is willing to wait for one of the specified events to occur.

#### Note:

1. Seconds can be any value greater than or equal to 0, and less than or equal to 4 294 967 295. The value specified for Seconds is an unsigned integer.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

### Nanoseconds

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains an unsigned integer that represents the number of nanoseconds to be added to the value that is specified by Seconds.

#### Note:

1. Nanoseconds can be any value greater than or equal to 0, and less than 1 000 000 000.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the service returns 0 if a CW\_CONDVVAR event occurred, or -1 if it has not.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the `set_timer_event` service stores the return code. The `set_timer_event` service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The `set_timer_event` service can return one of the following values in the `Return_code` parameter:

<b>Return_Code</b>	<b>Explanation</b>
<b>EINVAL</b>	One or more of the parameters that were passed to the service are in error. The following reason codes unique to the <code>set_timer_event</code> can accompany the return code: <code>JRNanoSecondsTooBig</code> , <code>JRBadPET</code> , <code>JrReleasedPET</code> .

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the `set_timer_event` service stores the reason code. The `set_timer_event` service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Once the time has expired, the kernel posts the ECB that is located at `ThliTimerEcb`, mapped by `BPXYTHLI`. The kernel clears this ECB before the timer is set.
2. The timer is canceled on the next syscall, or if the thread is terminated.
3. If the timer is set to a small enough value, the ECB that is defined at location `ThliTimerEcb` may already have been posted before control is returned to the caller.
4. If a valid unauthorized PET is stored in `ThliPET` before the `BPX1STE/BPX4STE` call, the kernel will `RELEASE (IEAVRLS)` the PE associated with `ThliPET` instead of posting `ThliTimerEcb`. The invoker of the `PAUSE (IEAVPSE)` will receive a release code of `Thli#PauseTimeout`.

**Related services**

- [“cond\\_timed\\_wait \(BPX1CTW, BPX4CTW\) — Suspend a thread for a limited time or an event”](#) on page 132

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1STE \(set\\_timer\\_event\) example”](#) on page 1297.

## setuid (BPX1SUI, BPX4SUI) — Set user IDs

---

**Function**

The `setuid` callable service sets the real, effective, and saved set user IDs for the current process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SUI):</b>	31-bit
<b>AMODE (BPX4SUI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SUI, (User_ID,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SUI with the same parameters.

## Parameters

### User\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the user ID the process is to assume.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setuid service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the setuid service stores the return code. The setuid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in

*z/OS UNIX System Services Messages and Codes* The setuid service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EINVAL</b>	The user ID that was specified was incorrect.
<b>EMVSSAF2ERR</b>	The SAF call IRRSSU00 incurred an error.
<b>EPERM</b>	The process does not have the appropriate privileges to set the user ID. Refer to <a href="#">“Authorization” on page 8</a> for information on appropriate privileges.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the setuid service stores the reason code. The setuid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see Reason codes in *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETUID service return and reason code values, see the following table:

<b>RACF return code</b>	<b>RACF reason code</b>	<b>Explanation</b>
<b>8</b>	<b>4</b>	UID is not defined to RACF
<b>8</b>	<b>8</b>	User not authorized to change UID
<b>8</b>	<b>12</b>	Internal error during RACF processing
<b>8</b>	<b>16</b>	Unable to establish recovery

For a more detailed description of the RACF CKPRIV service return and reason code values, see the following table:

<b>RACF return code</b>	<b>RACF reason code</b>	<b>Explanation</b>
<b>8</b>	<b>4</b>	User is not privileged
<b>8</b>	<b>12</b>	Internal error during RACF processing

**Usage notes for setuid**

1. If a UNIX set-user-ID privileged program switches its UID to other than that of the set-user-ID program and it causes the real, effective and saved UID of the caller to be equal, then the set-user-ID privilege of the program is given up.
2. If User\_ID is the same as the real UID of the process or the saved set UID, the setuid service sets the effective UID to be the same as User\_ID.
3. If User\_ID is not the same as the real UID of the process, and the calling process has appropriate privileges, the real, effective, and saved set UIDs are set to User\_ID. For more information about appropriate privileges, see [“Authorization” on page 8](#).
4. In z/OS UNIX, you change the identity of a process by changing the real and effective UIDs and the supplementary groups. In order to change the identity of the process on MVS, you must also change the MVS security environment. The setuid function calls SAF services to change the MVS identity (user ID) of the address space to the user ID that is associated with the target UID only if the caller



is a daemon, or if the target user ID has been properly authenticated. If the caller is a daemon, the following conditions must be met:

- The caller must be a superuser (UID=0).
- If the BPX.DAEMON profile is defined in the FACILITY class, the caller must be permitted to this profile.
- The calling program must be loaded from a controlled library, as defined by the RACF support for program access to data sets (PADS). For more information, see [seesteps for setting up enhanced program security in z/OS UNIX System Services Planning](#).

If the caller is not a daemon, the target user ID must have been authenticated in one of the following ways:

- a. Successful completion of the password service, where the user ID specified is associated with the target UID of the setuid service.
  - b. If the caller of the setuid service has read access to the BPX.SRV.userid profile in the SURROGATE class, where *userid* is the user ID that is associated with the target UID, permission is granted to perform the specified operation. For more information about setting up surrogate profiles, see [Defining servers to process users without passwords in z/OS UNIX System Services Planning](#).
5. When the MVS identity is to be changed, the target MVS user ID is determined as follows:
    - If an MVS user ID is already known (saved) by the kernel from a previous call to the getpwnam or the password service calls, and the UID created for this user ID matches the UID that is specified on the setuid call, this saved user ID is used.
    - For nonzero target UIDs, if there is no known user ID, or if the UID for the known user ID does not match the UID that is requested on the setuid call, the setuid service queries the security database to retrieve the user ID. The retrieved user ID is then used.
    - If the target UID is 0 and a user ID is not known, the setuid service sets the MVS user ID to BPXROOT, or to a user ID that is specified as a parmlib option during installation. BPXROOT is set up during system initialization as a superuser with a UID of 0. The BPXROOT user ID is not defined to the BPX.DAEMON profile in the FACILITY class. This special processing is necessary to prevent a superuser from gaining daemon authority.
    - When a change is being made from a nonzero UID to a zero UID, the MVS user ID is not changed. When you use the su shell command to become a superuser, your shell retains your original MVS user ID.
  6. The MVS identity is not changed on a successful call to setuid in the following situations:
    - When a change is being made from a nonzero UID to a zero UID. When you use the su shell command to become a superuser, your shell retains your original MVS user ID.
    - When it is running in a setuid program, and a setuid is done back to the original real UID.
  7. You should be careful when you are constructing the MVS identity associated with a setuid program. These programs effectively allow a subsequently spawned child non-setuid program to set its effective UID and associated MVS identity to the UID and MVS identity of the setuid of the program.
  8. The setuid service is not supported from an address space that is running multiple processes, because it would cause all processes in the address space to have their security environments changed unexpectedly. The call to the setuid service in this environment fails with an EMVSERR return code and a JRMultiProc reason code.
  9. The setuid service is not supported from a TSO address space. The call to the setuid service in this environment fails with an EMVSERR return code and a JRTso reason code.
  10. The setuid service is not supported from a task that is running with a previously obtained task-level security environment. The call to the setuid service in this environment fails with an EMVSERR return code and a JRTaskAcee reason code.
  11. The setuid service is not supported in the following situation: The BPX.DAEMON profile is defined in the FACILITY class and the caller is attempting to change its security environment by changing its MVS user identity, but a load was issued from an uncontrolled data set in the caller's address space.

This address space could be corrupted; for this reason, daemon activity is not allowed. The call to the `setuid` service in this environment fails with an `EMVSERR` return code and a `JREnvDirty` reason code.

12. The `setuid` service is not supported when running from within a `setuid` program, because in most cases the MVS identity will not change in this environment.
13. To attach the security environment of the caller of the `setuid` service to the security environment of the target UID (which then creates a nested ACEE for the target), use the `_BPXK_DAEMON_ATTACH` environment variable. The new client can then access RACF delegated resources for which the daemon, but not necessarily the client, has access. (The delegated resources are designated by the `APPDATA` text of 'RACF-DELEGATED' in the RACF profile protecting the resource.)
14. The `_BPXK_SUID_FORK` environment variable specifies whether the `setuid` indicator is propagated to child address spaces created by the fork service. For more information, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).
15. When the effective UID is changed, the MVS identity is also changed. The original MVS identity (ACEE) is saved so that the address space can be restored to its original security environment when the process terminates. During process termination, no other tasks in the address space should make any security calls since they cannot depend on the address space security environment and unpredictable results may occur.

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“geteuid \(BPX1GEU, BPX4GEU\) — Get the effective user ID” on page 264](#)
- [“getuid \(BPX1GUI, BPX4GUI\) — Get the real user ID ” on page 329](#)
- [“seteuid \(BPX1SEU, BPX4SEU\) — Set the effective user ID” on page 760](#)
- [“setgid \(BPX1SGI, BPX4SGI\) — Set the group ID” on page 762](#)

## Characteristics and restrictions

If the `setuid` service is used within a multi-threaded process, use synchronization to ensure that the `setuid` service is not performed concurrently with other z/OS UNIX services. Unserialized use can yield unpredictable results.

See also [“Usage notes for `setuid`” on page 802](#).

## Examples

See [“BPX1SUI \(setuid\) example” on page 1299](#).

# shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment

---

## Function

The `shmat` service attaches the shared memory segment that is associated with a shared memory identifier.

Amode 31:

- Each `shmat()` to the same shared memory segment results in a storage OBTAIN in the user address space. A unique storage address is returned from `shmat()`.
- A display of attaches via the shell IPCS command will show each unique attach.

Amode 64:

- Each `shmat()` to the same shared memory object is pointing to the same storage. The same storage address is returned from `shmat()`.

- When there are multiple attachers to the same memory object from within a process, the shell IPCS command shows just one attach for the object.
- A maximum of 1000 attaches to the same memory object from a single process is allowed.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state; PSW key 2, 8, or 9
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MAT):</b>	31-bit
<b>AMODE (BPX4MAT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MAT, (Shared_Memory_ID,
               Shared_Memory_Address,
               Shared_Memory_Flag,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers need an additional parameter:

```
CALL BPX4MAT, (Shared_Memory_ID,
               Shared_Memory_Address,
               Shared_Memory_Flag,
               Attached_Address,
               Return_value,
               Return_code,
               Reason_code)
```

The Shared\_Memory\_Address parameter is a doubleword.

## Parameters

### Shared\_Memory\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Specifies the shared memory identifier that is returned by the shmget service.

### Shared\_Memory\_Address

Supplied parameter

#### Type:

Address

**Length:**

Fullword (doubleword)

For BPX1MAT, the name of a field that contains a 31-bit address of where in the caller's address space storage is to be obtained and the segment is to be attached, or a 31-bit zero.

For BPX4MAT, the name of a field that contains a 64-bit address of where in the caller's address space storage is to be obtained and the segment is to be attached, or a 64-bit zero.

- If Shared\_Memory\_Address is a null pointer, the segment is attached at the first available address selected by the system that is on a page boundary; or on a megabyte boundary, if the shared memory segment is defined as an IPC\_MEGA segment.
- If Shared\_Memory\_Address is not a null pointer and Shm\_RND is specified, the segment's storage address is truncated to a page boundary (last 12 bits zero); or to a megabyte boundary (last 20 bits zero), if the shared memory segment is defined as an IPC\_MEGA segment.
- If Shared\_Memory\_Address is not a null pointer and Shm\_RND is not specified, the segment is attached at the address that is specified. If the shared memory segment is defined as an IPC\_MEGA segment, the specified address must be a megabyte multiple, or the request is failed with an EINVAL.

For BPX4MAT, the address must always be the same, and it must be above the bar.

If the shared memory segment is defined as an IPC\_MEGA segment, the value that is specified in Shared\_Memory\_Address must be either zero or equal to or greater than 16 megabytes; otherwise, the request fails with an EINVAL.

If the segment being attached is above the 2G bar (that is, it is a 64-bit address), the Shared\_Memory\_Address must either be zero or the same address that was returned on the shmgt() call. If the segment being attached is below the bar (that is, it was created with option IPC\_BELOWBAR or IPC\_MEGA on the BPX4MGT call), then the Shared\_Memory\_Address follows the same rules as for BPX1MAT callers.

**Shared\_Memory\_Flag**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

Shm\_RDONLY identifies the segment that is to be attached for read only; otherwise, the segment is attached for read and write. Shm\_RDONLY has no effect for attaches to shared memory segments that are created with the IPC\_MEGA option. Whether the segment is attached read only or read and write depends on how it is currently accessed by other attaches, as all users have the same access to shared memory that is created with the IPC\_MEGA option.

Shm\_RND causes the Shared\_Memory\_Address to be truncated to a page boundary (last 12 bits zero), or to a megabyte boundary (last 20 bits zero) if the shared memory segment is defined as an IPC\_MEGA segment.

**Attached\_Address**

Returned parameter (BPX4MAT only)

**Type:**

Integer

**Length:**

Doubleword

The name of a doubleword in which the shmat service returns the shared memory segment address (the address that is to be passed to the detach) when Return\_value is zero.

**Return\_value**

Returned parameter

**Type:**

Address

**Length:**

Fullword

The name of a fullword in which the shmat service returns the shared memory segment address (the address that is to be passed to the detach), or -1, if the operation is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmat service stores the return code. The shmat service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The shmat service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EACCES**

One of the following conditions occurred:

- Operation permission is denied to the caller. The combination of Shared\_Memory\_Flag and permissions denies the requester access. The following reason code can accompany the return code: JRIPcDenied.
- The caller's PSW key does not match the key of the shared memory segment (except that callers running in PSW key 8 can attach to a key 9 shared memory segment). The following reason code can accompany the return code: JrKeyMismatch.

**EINVAL**

Shared\_Memory\_ID is not a valid shared memory identifier, for one of the following reasons:

- Shared\_Memory\_Address is not zero, it is not on a page boundary, and SHM\_RND was not specified.
- Shared\_Memory\_Address is not zero, it is not on a megabyte boundary, and SHM\_RND was not specified.
- The storage at Shared\_Memory\_Address could not be obtained in the user's address space.
- The caller is not running with a PSW key of 2, 8, or 9.

The following reason codes can accompany the return code: JRIPcBadID, JRBadAddress, JRNoUserStorage, JRStorNotAvail, or JrUnsupportedKey.

**EMFILE**

The number of shared memory segments attached to the caller's process exceeds the system-imposed maximum. This system limit is set with the IPCSHMNSEGS parameter in a BPXPRMxx parmlib member. You can use the `ipcs -x` shell command to view this value. The following reason code can accompany the return code: JRShmMaxAttach.

**ENOMEM**

The available system storage is not large enough to accommodate the shared memory segment. The following reason codes can accompany the return code: JRNoUserStorage, JRSMNoStorage, JRlarvserv or JRShrStgShortage.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmat service stores the reason code. The shmat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If an attempt is made to access memory outside the shared memory segment, normal address space storage is accessed.
2. It is the application's responsibility to determine the length of the shared memory segment that is attached.
3. If an attempt is made to update a shared memory segment that is attached with Shm\_RDONLY access, a program check occurs.
4. Because of the nature of the mapping of shared memory segments to different addresses within the multiple processes it is attached to, relative addresses should be used as pointers within the shared memory segment.
5. The storage is allocated in subpool 129, which is associated with the job step task. This allows a thread to attach a shared memory segment and exit, allowing other threads in the process to access the storage.
6. See [“shmget \(BPX1MGT, BPX4MGT\) — Create/find a shared memory segment” on page 827](#), usage note “10” on page 831, for an explanation of the storage key chosen for a shared memory segment.
7. Above the bar, shared memory cannot be used in subspace mode.

**Related services**

- [“shmctl \(BPX1MCT, BPX4MCT\) — Perform shared memory control operations ” on page 809](#)
- [“shmdt \(BPX1MDT, BPX4MDT\) — Detach a shared memory segment” on page 812](#)
- [“shmget \(BPX1MGT, BPX4MGT\) — Create/find a shared memory segment” on page 827](#)

**Characteristics and restrictions**

- The invoker is restricted by ownership and read and write permissions defined by shmget and shmctl IPC\_SET.
- Restricted to callers running in PSW key 2, 8, or 9. Authorized users can exploit the IARVSERV macro directly to create shared memory in system keys.

**Examples**

See [“BPX1MAT \(shmat\) example” on page 1259](#).

## shmctl (BPX1MCT, BPX4MCT) – Perform shared memory control operations

---

### Function

The shmctl service provides a variety of shared memory control operations as specified by the Command parameter. These functions include reading and changing shared memory variables with the shmid\_ds data structure, and removing a shared memory segment from the system.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state; PSW key 2, 8, or 9
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MCT):</b>	31-bit
<b>AMODE (BPX4MCT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1MCT, (Shared_Memory_ID,
               Command,
               Buffer_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MCT with the same parameters. The Buffer\_address parameter is a doubleword.

### Parameters

#### Shared\_Memory\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Specifies the shared memory identifier.

#### Command

Supplied parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword field that indicates the shared memory command that is to be executed. For the structure that contains these constants, see [“BPXYSHM—Map interprocess communication shared memory segments”](#) on page 1147 and [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098. The values for Command are:

**Ipc\_STAT**

This command obtains status information about the shared memory that is identified by the Shared\_Memory\_ID parameter, if the current process has read permission. This information is stored in the area that is pointed to by the Buffer\_address parameter and mapped by area SHMID\_DS data structure. For the data structure, see [“BPXYSHM—Map interprocess communication shared memory segments”](#) on page 1147, SHMID\_DS DSECT.

**Ipc\_STAT\_EX**

When the AMODE of the caller is 31, it provides the same output as IPC\_STAT when AMODE is 64 (that is, extended POSIX times are provided). When the AMODE of the caller is 64, IPC\_STAT\_EX and IPC\_STAT are equivalent.

**Ipc\_SET**

This command sets the value of the IPC\_UID, IPC\_GID and IPC\_MODE from the SHMID\_DS data structure that is associated with Shared\_Memory\_ID into the SHMID\_DS structure that is pointed to by the Buffer\_address parameter. Any value for IPC\_UID and IPC\_GID can be specified. Only mode bits that are defined by semget under Semaphore\_Flag argument can be specified in the IPC\_MODE field. This command can only be executed by a process with an effective user ID equal to either that of a process with appropriate privileges (see [“Authorization”](#) on page 8), or to the value of IPC\_CUID or IPC\_UID in the SHMID\_DS data structure that is associated with Shared\_Memory\_ID. This information is taken from the buffer pointed to by the Buffer\_address parameter. For the data structure, see [“BPXYSHM—Map interprocess communication shared memory segments”](#) on page 1147, SHMID\_DS DSECT.

For shared memory segments that were not created with the Ipc\_MEGA option, the permissions that are in effect (IPC\_MODE) when a process attaches a segment remain, even though these permissions might change. For shared memory segments that were created with the Ipc\_MEGA option, the permissions that are set by this request take effect immediately. All processes that are currently attached to the shared memory segment are able to read only or read and write to it based on the permissions that are specified in the IPC\_MODE.

The effect of the new mode on access is determined by the three parts of the mode field: the owner permissions, the group permissions, and other permissions. If all three read and all three write permissions in the new mode are off, the access for all attached processes is changed to read. If any of the three read permission bits is on without the corresponding write permission bit on, the access for all attached processes is changed to read. Otherwise, the access for all attached processes is changed to write.

**Ipc\_SET\_EX**

When the AMODE of the caller is 31, it assumes the same input as IPC\_SET when AMODE is 31 and provides the same output as IPC\_SET when AMODE is 64 (that is, extended POSIX times are provided). When the AMODE of the caller is 64, IPC\_SET\_EX and IPC\_SET are equivalent.

**Ipc\_RMID**

This command removes the shared memory identifier that is specified by Shared\_Memory\_ID from the system, and removes the shared memory segment and SHMID\_DS data structure that are associated with it. This command can only be executed by a process with an effective user ID equal to either that of a process with appropriate privileges (see [“Authorization”](#) on page 8), or to the value of IPC\_CUID or IPC\_UID in the SHMID\_DS data structure that is associated with Shared\_Memory\_ID.

Users who have READ access to the SUPERUSER.IPC.RMID profile in the UNIXPRIV class are treated as superusers.



**Buffer\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the buffer that is mapped by SHMID\_DS. The shmctl service assumes that the size of this buffer is at least as large as SHMID\_DS.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmctl service stores the return value, or a -1 if the operation is unsuccessful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmctl service stores the return code. The shmctl service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The shmctl service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The command specified was Ipc_STAT, and the calling process does not have read permission. The following reason code can accompany the return code: JRIPcDenied.
<b>EFAULT</b>	The Buffer_Address parameter specified an address that caused the callable service to program check. The following reason code can accompany the return code: JRBadAddress.
<b>EINVAL</b>	This error code is returned for the following reasons: <ul style="list-style-type: none"> <li>• Shared_Memory_ID is not a valid shared memory identifier.</li> <li>• The Command parameter is not a valid command.</li> <li>• The mode bits were not valid (SET). The following reason codes can accompany the return code: JRIPcBadFlags, JRIPcBadID and JRBadEntryCode.</li> </ul>
<b>EPERM</b>	Command=IPC_RMID or IPC_SET, and the effective user ID of the caller is not that of a process with appropriate privileges (see <a href="#">“Authorization” on page 8</a> ), and is not the value of IPC_CUID or IPC_UID in the SHMID_DS data structure that is associated with Shared_Memory_ID. The following reason code can accompany the return code: JRIPcDenied.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmctl service stores the reason code. The shmctl service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

1. The remove operation is asynchronous to the return from the system call after the last attachment is broken.
2. When a RMID is processed, no further attaches are allowed.
3. Ipc\_SET can change permissions, and might affect a thread's ability to use the shared memory functions.
4. If an RMID was processed before a fork service, the child is not attached to the memory segment.
5. Above the bar, shared memory cannot be used in subspace mode.

**Related services**

- [“w\\_getipc \(BPX1GET, BPX4GET\) — Query interprocess communications” on page 991](#)
- [“shmat \(BPX1MAT, BPX4MAT\) — Attach to a shared memory segment ” on page 804](#)
- [“shmdt \(BPX1MDT, BPX4MDT\) — Detach a shared memory segment” on page 812](#)
- [“shmget \(BPX1MGT, BPX4MGT\) — Create/find a shared memory segment” on page 827](#)

**Characteristics and restrictions**

The caller of the shmctl service is restricted by ownership and read and read/write permissions that are defined by shmget and shmctl Ipc\_SET.

**Examples**

See [“BPX1MCT \(shmctl\) example” on page 1259](#).

## **shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment**

---

**Function**

The shmdt service detaches a shared memory segment.

When there are multiple attachers to the same 64 bit memory object from within a process, the shmdt will not detach the storage from the process until all attachers from within the process have issued a detach.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state; PSW key 2, 8, or 9
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MDT):</b>	31-bit
<b>AMODE (BPX4MDT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts

**Operation****Environment****Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1MDT, (Shared_Memory_Address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MDT with the same parameters. The Shared\_Memory\_Address parameter is a doubleword.

**Parameters****Shared\_Memory\_Address**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the starting address of a shared memory segment. This is the Return\_value from shmat (BPX1MAT,BPX4MAT). The address returned is 31 bits for AMODE 31 callers and 64 bits for AMODE 64 callers.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmdt service returns 0 if the request was successful, or -1 if the operation was unsuccessful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmdt service stores the return code. The shmdt service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The shmdt service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EINVAL**

Shared\_Memory\_Address is not the data segment start address of a shared memory segment attached to the caller's process. The following reason code can accompany the return code: JRBadAddress.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmdt service stores the reason code. The shmdt service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- “shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment ” on page 804
- “shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations ” on page 809
- “shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment” on page 827

**Usage notes**

- When a shared memory segment is detached via shmctl(), the process loses access to the associated shared memory. If this is done before a process has finished using all condition variables and mutexes in the detached shared memory segment, unpredictable results will occur in the application, leading to possible hangs and loss of resources. Because a process is implicitly detached from its shared memory attachments when it terminates, most applications should avoid doing any explicit shmdt() calls before terminating.
- Above the bar shared memory cannot be used in subspace mode.
- For a segment of type IPC\_SHAREAS, a detach call will not cause the cleanup of the segment's storage within the user address space unless no other processes in the address space are attached to the segment.

**Characteristics and restrictions**

The caller of the shmdt service is restricted by ownership and read and read-write permissions that are defined by shmget and shmctl Ipc\_SET.

**Examples**

See “BPX1MDT (shmdt) example” on page 1260.

## **shmem\_lock (BPX1SLK, BPX4SLK) — Shared memory lock service**

---

**Function**

The shmem\_lock callable service provides a general-purpose interface for managing and operating locks in shared memory. It allows an application to serialize resources that must be shared across multiple address spaces.

**Requirements****Operation****Environment****Authorization:**

Supervisor state or problem state, any PSW key

**Dispatchable unit mode:**

Task

**Cross memory mode:**

PASN = HASN

Operation	Environment
<b>AMODE (BPX1SLK):</b>	31-bit
<b>AMODE (BPX4SLK):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SLK, (LockFcnCode,
               LockReqType,
               LockType,
               LockAddr,
               LockAttrAddr,
               LockTokenAddr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SLK with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### LockFcnCode

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value indicating the function requested. The following are the supported values:

#### SLK\_INIT

A new shared memory lock is to be created and initialized.

#### SLK\_DESTROY

A shared memory lock is to be destroyed and its resources cleaned up.

#### SLK\_OBTAIN

A shared memory lock is to be obtained unconditionally.

#### SLK\_OBTAIN\_COND

A shared memory lock is to be obtained on the condition that is not already obtained. If the requested lock is not available immediately, the request will fail (EBUSY) without blocking.

#### SLK\_RELEASE

A shared memory lock is to be released.

These constants are defined in the BPXYCONS macro ([“BPXYCONS — Constants used by services” on page 1062](#)).

### LockReqType

Supplied parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword that contains one or more of the following values indicating the lock request type. This parameter is valid only for the SLK\_INIT function, and is ignored for all other functions. The following are the supported values:

**SLK\_NORMAL**

A new shared memory lock is to be created with no deadlock detection. This value is mutually exclusive with the SLK\_ERRORCHECK value.

**SLK\_ERRORCHECK**

A new shared memory lock is to be created with deadlock detection. This value is mutually exclusive with the SLK\_NORMAL value.

**SLK\_RECURSIVE**

A new shared memory lock is to be created with a recursive locking capability. This allows the same lock to be obtained multiple times by the same caller, without requiring intervening releases and without causing deadlock. To take advantage of this capability, the lock must be obtained with the same lock type on each obtain call.

These constants are defined in the BPXYCONS macro ([“BPXYCONS — Constants used by services”](#) on page 1062).

**LockType**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains one or more of the following values indicating the lock type. This parameter is valid only for the SLK\_INIT and SLK\_OBTAIN functions; it is ignored for all other functions. For the SLK\_INIT function, the request type values can be combined to create a multiple-type lock (that is, a lock that can be obtained either shared or exclusively). For the SLK\_OBTAIN function, only one of the values can be specified on a given call. The following are the supported values:

**SLK\_SHARED**

A shared memory lock is to be created or obtained with the shared attribute. A lock that is obtained with the shared attribute can be obtained concurrently by other callers requesting a shared lock obtain. A lock initialized with this value is, by default, defined as a recursive lock.

**SLK\_EXCLUSIVE**

A shared memory lock is to be created or obtained with the exclusive attribute. A lock that is obtained with the exclusive attribute cannot be obtained concurrently by other callers.

These constants are defined in the BPXYCONS macro; see [“BPXYCONS — Constants used by services”](#) on page 1062.

**LockAddr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user lockword in shared memory.

**LockAttrAddr**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the lock attribute area. The LockAttrAddr parameter is for use with the SLK\_INIT function only. It is intended to allow for potential extensions to the shared memory locks. Because these extensions are not currently supported, the caller of the shmem\_lock service should specify a null pointer for the lock attribute area address.

**LockTokenAddr**

Supplied and returned parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a fullword that the service uses either to return a lock token, or as the input lock token. When it is specified with the SLK\_INIT function, the LockTokenAddr parameter is used as the address of an output area in which to return the lock token of the newly created lock. For all other functions, this parameter contains the address of the lock token that represents the lock to be operated upon.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmem\_lock service returns 0, if the request is successful; or -1, if it is not successful. For all successful SLK\_INIT and SLK\_DESTROY function requests, the shmem\_lock service returns 0. For successful SLK\_OBTAIN, SLK\_OBTAIN\_COND, and SLK\_RELEASE function requests, the shmem\_lock service returns a count of the number of times the calling thread has had the requested lock held.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmem\_lock service stores the return code. The shmem\_lock service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The shmem\_lock service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	The requested service could not be performed at the current time because of a lack of available system resources. The following reason codes can accompany this return code: JRTLOCKMAXCNTTHD, JRLOCKMAXCNTSYS, JRLOCKMAXCNTRECURSE.
<b>ENOMEN</b>	The requested service could not be performed at the current time because of a lack of available system storage.
<b>EINTR</b>	A signal interrupted the callable service.

Return_code	Explanation
<b>EINVAL</b>	One of the parameters contains a value that is not correct. The following reason codes can accompany this return code: JRLOCKFCNCODE, JRLOCKREQTYPE, JRLOCKTYPE, JRLOCKADDR, JRLOCKTOKEN.
<b>EFAULT</b>	One of the parameters contains an address that is not accessible by the caller. The following reason code can accompany the return code: JRLOCKTOKENADDR.
<b>EBUSY</b>	The specified function cannot be performed because a required resource is already in use. The following reason codes can accompany the return code: JRLOCKINUSE, JRLOCKEDALREADY.
<b>EPERM</b>	The caller is not permitted to perform the specified operation. The following reason codes can accompany the return code: JRLOCKNOTOWNER, JRLOCKSHMACC.
<b>EDEADLK</b>	The caller already owns the lock that is requested.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmem\_lock service stores the reason code. The shmem\_lock service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

## 1. Lock initialization

In order for a lock initialization call to complete successfully, the specified lock address must be in a memory-mapped area or shared storage segment that is read-write accessible by the calling process. To most efficiently manipulate locks in shared storage, it is recommended that the lock be in a shared memory segment. The shmem\_lock service is optimized to handle locks that reside in a shared memory segment, rather than a memory-mapped area.

A successful lock initialization call causes a lock token representing the newly created lock to be returned in the lock token output area that is supplied via the LockTokenAddr parameter. Subsequent calls to the shmem\_lock service to manipulate the newly created lock must specify the returned lock token in order to identify the lock that is to be manipulated. A lock initialization call can fail if system resources other than storage are not available to initialize the lock (EAGAIN), or if not enough system storage is available (ENOMEM).

## 2. Lock destroy

A destroy of a lock causes the system resources for that lock to be cleaned up, if the lock is not in use. If the lock is in use, the destroy request fails (EBUSY). A lock could be in use if a thread has it in a locked state, or if it is being referenced by another thread on a pthread\_cond\_timedwait or pthread\_cond\_wait. Once a lock is destroyed, any further operations against that lock fail (EINVAL).

## 3. Lock obtain

A successful call to the shmem\_lock service to obtain a shared memory lock results in a GRS latch obtain against a latch in the 'SYS.BPX.AP00.GXSLT.SHMLOCKS.LSN' latch set. If an application is experiencing serialization problems with a shared memory lock, GRS contention analysis tools such as D GRS,C and IPCS ANALYZE can be used to determine the cause of the problem. The lower halfword



of the lock token that is returned by the `shmem_lock` service indicates the latch number of the corresponding latch within the 'SYS.BPX.AP00.GXSLT.SHMLOCKS.LSN' LATCH set.

If an exclusive obtain of a lock that is defined as both exclusive and shared is attempted by a thread that already has that lock obtained exclusively, deadlock results. Additionally, if an exclusive or shared obtain of a shared and exclusive lock is attempted by a thread that already has that lock obtained exclusively, deadlock results. To prevent exclusive obtain starvation for a lock that is defined as shared and exclusive, a new shared lock obtain blocks if there are any exclusive obtain callers waiting. A lock that is initialized with the recursive attribute can be obtained multiple times by the same thread, up to a limit of 32 768 iterations. A single thread can hold up to a limit of 128 different shared memory locks concurrently.

#### 4. Lock release

A lock release call against a lock that is not in a locked state or that is not owned by the calling thread results in an error (EPERM). A lock with the recursive attribute that has been obtained  $n$  times by a given thread must be released  $n$  times by that same thread in order for the lock to be completely released.

#### 5. System cleanup

During task termination processing of a thread that ends while it is holding a shared memory lock, the lock is released by the system. If a job step ends abnormally (for example, if it is canceled), or if an address space is terminated at end of memory, all shared memory locks that are held by that job or address space are released.

### Related services

None.

### Characteristics and restrictions

None.

### Examples

See [“BPX1SLK \(shmem\\_lock\) example”](#) on page 1289.

## shmem\_mutex\_condvar (BPX1SMC, BPX4SMC) – Shared mutex and condition variable service

---

### Function

The `shmem_mutex_condvar` callable service provides a general-purpose interface for managing and operating mutexes and condition variables in shared memory. An application can:

- Create and initialize a shared memory mutex or condition variable.
- Destroy a shared memory mutex or condition variable and clean up its resources.
- Post the oldest waiter for a specified mutex or condition variable.
- Post all of the waiters for a specified mutex or condition variable.
- Wait for a specified condition variable.
- Setup to wait for a mutex.
- Cancel setup to wait for a specified mutex.
- Wait for a specified condition variable and post any waiters for the specified mutex.

## Requirements

### Operation

#### Authorization:

Supervisor state or problem state, any PSW key

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1SMC):

31-bit

#### AMODE (BPX4SMC):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SMC, (FcnCode,
               FcnFlags,
               ShrObj1Addr,
               ShrObj2Addr,
               EcbAddr,
               TimeStrucAddr,
               UserDataAddr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SMC with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### FcnCode

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value indicating the function requested. The following values are supported:

#### SMC\_INIT

Create and initialize a new shared memory mutex or condition variable.

#### SMC\_DESTROY

Destroy a shared memory mutex or condition variable and clean up its resources.

#### SMC\_POST

Post the oldest waiter for the specified mutex or condition variable.

#### SMC\_POSTALL

Post all of the waiters for the specified mutex or condition variable.

#### SMC\_WAIT

Wait for the specified condition variable.

**SMC\_SETUPWAIT**

Setup to wait for a mutex. This function is not supported for condition variables.

**SMC\_CANCELSETUPWAIT**

Cancel set up to wait for the specified mutex or condition variable.

**SMC\_WAIT+SMC\_POST**

Wait for the specified condition variable and post any waiters for the specified mutex.

These constants are defined in the BPXYCONS macro ([“BPXYCONS — Constants used by services”](#) on page 1062).

**FcnFlags**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the function flags for the requested function. The possible values for FcnFlags are:

**SMC\_Mutex**

The input shared memory object represents a mutex. This setting must be specified on all function calls that involve a mutex object.

**SMC\_Condvar**

The input shared memory object represents a condition variable. This setting must be specified on all calls that involve a condition variable. SMC\_Condvar or SMC\_Mutex must be specified on all calls to BPX1SMC/BPX4SMC.

**SMC\_TimedWait**

Wait for a specified time interval for the specified condition variable. This option is relevant only for a condition variable. When SMC\_TimedWait is specified, the TimeStrucAddr parameter points to the time structure that indicates the amount of time to wait.

**SMC\_OutsideWait**

The caller will wait outside of the BPX1SMC/BPX4SMC function for the specified mutex or condition variable. This setting is relevant only for the SMC\_SetupToWait function. SMC\_OutsideWait must be specified when SMC\_Mutex is specified.

**ShrObj1Addr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the shared condition variable or mutex control structure (SMCV or SMMX) that is involved in the specified operation. If SMC\_MUTEX is specified in the FcnFlags, this parameter must point to a valid shared mutex control structure (SMMX). If SMC\_CONDVAR is specified in the FcnFlags, this parameter must point to a valid shared condition variable control structure (SMCV).

**ShrObj2Addr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the mutex control structure (SMMX) that is associated with the supplied shared condition variable structure (SMCV).

When SMC\_CONDVAR is specified in the FcnFlags for a SMC\_Wait or SMC\_SetupToWait function request, this parameter must contain a pointer to a valid SMMX. For all other calls to this service, this parameter must be specified, but its value will not be validity checked.

**EcbAddr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the ECB that the caller will wait on for the specified mutex or condition variable. For the SMC\_SetupToWait function for a mutex object, this parameter must contain a valid ECB address. For all other calls to this service, this parameter must be specified, but its value will not be validity checked. For both BPX1SMC and BPX4SMC callers, the ECB must be in below-the-bar storage.

**TimeStrucAddr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the time structure (SMCT) that indicates the amount of time the caller will wait for a specified condition variable. When the SMC\_CondVar and SMC\_TimedWait function flags are specified with the SMC\_SetupToWait or SMC\_Wait function, this parameter must point to a valid SMCT structure. For all other calls to this service, this parameter must be specified, but its value will not be validity checked.

**UserDataAddr**

Supplied parameter

**Type:**

Pointer

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of user data supplied by the caller for problem determination support. This parameter must be specified, but its value will not be validity checked.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the BPX1SMC/BPX4SMC service returns 0 if the request is successful, or -1 if it is not successful. For a successful initialization call (SMC\_INIT), the input shared memory object is filled in as appropriate for an SMMX (mutex) or SMCV (condition variable).

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the BPX1SMC/BPX4SMC service stores the return code. The BPX1SMC/BPX4SMC service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The BPX1SMC/BPX4SMC service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EAGAIN</b>	The requested service could not be performed at the current time because of a lack of available system resources. The following reason codes can accompany this return code: JRSMCMaxCntSys, JRSMCMaxCntSeg.
<b>ETIMEDOUT</b>	The requested service reached the specified time-out interval.
<b>EINTR</b>	A signal interrupted the callable service. The following reason code can accompany this return code: JRSignalArrived.
<b>EINVAL</b>	One of the parameters contains a value that is not correct. The following reason codes can accompany this return code: JRSMCFcnCode, JRSMCFcnFlags, JRSMCWrongMutex, JRSMCNotMutex, JRSMCNotCondVar, JRSMCAreadySetUp, JRSMCNotShared, JRSMCUnusable, JRSMCMutexSetUp, JRSMCMemoryMap, JRSMCNotOwner, JRSMCDisabled.
<b>EFAULT</b>	One of the parameters contains an address that is not accessible by the caller. The following reason codes can accompany this return code: JRSMCObjAddr, JRSMCEcbAddr, JRSMCTimesTrAddr.
<b>EBUSY</b>	The initialization or destroy function cannot be performed because the specified object is already in use. The following reason codes can accompany this return code: JRSMCWaiters, JRSMCMutexLocked, JRSMCCondWaiters, JRSMCAreadyInit.
<b>EPERM</b>	The caller does not have the appropriate privilege to perform the operation. The following reason code can accompany this return code: JRSMCShMAcc.

#### **Reason\_code**

Returned parameter

##### **Type:**

Integer

##### **Length:**

Fullword

The name of a fullword in which the BPX1SMC/BPX4SMC service stores the reason code. The BPX1SMC/BPX4SMC service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## **Usage notes**

### 1. The initialization function (SMC\_INIT)

You initialize a mutex or condition variable by calling the initialization function (SMC\_INIT), supplying the shared memory address of the mutex or condition variable. A valid shared memory address is one that is within a shared memory segment that is read/write accessible to the calling process. If a call to initialize a mutex or condition variable supplies an object that is not in shared memory, the call fails with `Return_code` `EINVAL` and `Reason_code` `JRSMCNotShared`. In this case, the Language Environment `pthread_cond_init()` and `pthread_mutex_init()` functions treat the object as a non-shared object, because it is not accessible to any other process.

A mutex or condition variable is represented by two shared memory serialization structures that are maintained between Language Environment and the kernel: the shared memory mutex block (SMMX)

and the shared memory condvar block (SMCV). The C/Language Environment `pthread_mutex_init()` and `pthread_cond_init()` functions indicate on the initialization call to BPX1SMC/BPX4SMC the type of call that is being requested by setting the `FcnFlags` value to `SMC_Mutex` or `SMC_Convar`. The BPX1SMC/BPX4SMC initialization function initially fills in an SMMX or SMCV structure with information that is pertinent to the mutex or condition variable, and returns the associated kernel-shared memory serialization token in that structure. This kernel token is used by the kernel as the anchor to the kernel control structures for the object. The address of the SMMX and/or SMCV must be supplied on all subsequent calls to BPX1SMC/BPX4SMC for the waiting, posting and destroying of the shared memory mutex or condition variable that is associated with the SMMX or SMCV structure.

To prevent overuse of kernel resources, z/OS UNIX imposes limits on the number of shared mutex and condition variables that can be initialized for any one shared memory segment and on any given system. For applications running on behalf of unauthorized users the limits are 65,535 per shared memory segment and 131,072 for the system. For applications running on behalf of authorized users the limits are 4,194,304 (4M) per shared memory segment and 134,217,728 (128M) for the system. An application is authorized if it runs with UID 0 or if it runs with an effective UID that has READ access to the SUPERUSER.SHMMCV.LIMIT resource in the UNIXPRIV class. Each shared memory mutex or condition variable requires roughly 80 bytes of storage in the OMVS address space, so the use of large numbers of mutexes and condition variables causes a significant increase in the amount of storage used in the OMVS address space. Refer to the discussion of the SUPERUSER.SHMMCV.LIMIT resource in [Using UNIXPRIV class profiles](#) in *z/OS UNIX System Services Planning* for additional considerations.

## 2. The post function (SMC\_POST)

The post function is used during a `pthread_cond_signal()`, or when a mutex is unlocked (`pthread_mutex_unlock()`, `pthread_cond_wait()`, or `pthread_cond_timed_wait()`) and there are waiters for a condition variable or mutex. The post function wakes up the oldest waiter. The post call that is done from `pthread_mutex_unlock` is made after the lock that is associated with the mutex (SMMX) is released. If a post is done against a mutex or condition variable that has no waiters, the call will succeed, but a subsequent wait call will still block. In other words, pre-signalling of a condition variable cannot be done.

## 3. The postall function (SMC\_POSTALL)

The postall function is used during a `pthread_cond_broadcast()`; it causes all waiters for a condition variable to be awoken. When all waiters have woken up, they contend for the mutex that is associated with the specified condition variable.

## 4. The setup to wait function (SMC\_SETUPWAIT)

The setup to wait function is used during a `pthread_mutex_lock()` function when waiting is necessary. Whenever this function needs to block, it must be called prior to waiting outside the kernel. The `FcnFlags` input parameter indicates an outside wait (`SMC_OutsideWait`), and the `EcbAddr` parameter must point to the ECB that will be waited upon. If the `SMC_Timed_Wait` flag is on, the `TimeStrucAddr` parameter must point to a valid SMCT time structure that describes the amount of time required for the timed wait. The setup to wait function is supported only for mutex objects.

## 5. The wait function (SMC\_WAIT)

The wait function can be used by itself to wait for a condition variable, or it can be combined with the post function as a way to wait for a condition variable, at the same time posting any waiters for a mutex. The wait function cannot be used for mutexes, because mutex waits are not signal enabled.

## 6. The wait and post function (SMC\_WAIT+SMC\_POST)

The combined wait and post function is intended for use on a `pthread_cond_timedwait()` and `pthread_cond_wait()` as a way to minimize system overhead. When `SMC_WAIT+SMC_POST` is used, the `ShrObj1Addr` parameter must point to a valid SMCV, and the `ShrObj2Addr` parameter must point to a valid SMMX. The wait function internally performs a setup to wait for the condition variable, and then performs the post of the mutex. If a thread does a setup to wait function call followed by a wait function call for the same object, the wait function will fail.

As part of the post operation, the SMMX lock is updated with CDS to indicate that the caller no longer owns the mutex. After attempting the post of the mutex object, the condition variable is waited upon. After waking up from the condition variable wait, the mutex is reobtained by BPX1SMC/BPX4SMC. Because the mutex may not be available immediately, the service may have to wait for the mutex. The first time a condition variable is waited upon with an associated mutex, the condition variable is tied to the specified mutex for the life of the condition variable and mutex. No other mutex can be associated with the specified condition variable, and no other condition variable can be associated with that mutex, until the condition variable or mutex is destroyed.

#### 7. The cancel setup to wait function (SMC\_CANCELSETUPTOWAIT)

The cancel setup to wait function must be called any time a setup to wait is done and a wait for the resource is not performed. If a cancel setup to wait fails, the caller may have already been posted for the associated resource. This function would probably only be used if it were detected that a resource (such as a mutex) had become available without requiring a wait.

8. For an asynchronous signal delivered to the thread, or for thread cancelation processing of a thread that is blocked on a condition variable when the cancelability enable state of the thread is set to PTHREAD\_CANCEL\_DEFERRED, the BPX1SMC/BPX4SMC service unblocks the thread and returns to Language Environment with an EINTR return code. Language Environment handles this in the same way that it handles an EINTR returned from BPX1CWA. The EINTR return code is not surfaced to the C application.
9. The user data address that is supplied on the call to BPX1SMC/BPX4SMC is used by Language Environment to supply the stack address for the calling thread. This data is recorded for all waiters for a mutex or condition variable, and is displayed for each requestor of a mutex or condition variable on the D OMVS,SER report. When Language Environment successfully obtains a mutex on behalf of a caller, the SMMXOwnerData field is filled in with this information for the same purpose.
10. If BPX1SMC/BPX4SMC is asynchronously interrupted by an abnormal condition (such as a X'22' abend) during critical condition variable and/or mutex processing, the condition variable and/or mutex are invalidated and made unusable. Only a destroy of the condition variable and/or mutex can be performed on the object after the interrupt.
11. Shared memory data area structures: the SMMX and the SMCV

The SMMX (BPXYSMMX) and the SMCV (BPXYSMCV) data area structures represent two new C data types that are supported by the C RTL/Language Environment: the new larger pthread\_mutex\_t data type for shared-memory-resident mutexes, and the new larger pthread\_cond\_t data type for shared-memory-resident condition variables, respectively. These data types must be defined to be on a doubleword boundary that is enforced by the definitions of the data type.

The SMMX and the SMCV are architected for use between Language Environment and the kernel. So that Language Environment can distinguish between the various types of mutexes and condition variables it supports, the first 8 bytes of the SMMX and SMCV must have specific bits on.

- SMMX

To represent the new larger type of shared mutex, the SMMX must have bits 0, 31, 62, and 63 on in the first 8 bytes of the structure. The characters SMMX (X'E2D4D4D7") are chosen for the ID field (the first 4 bytes) so that they meet this requirement for bits 0 and 31, at the same time providing a representative eye-catcher for the structure. The second 4 bytes of the structure are a flag word (SmmxFlags), in which the low-order two flag bits must always be initialized to being on, and must never be moved from their bit positions.

Language Environment must serialize the use of a mutex that is represented by this structure, using CDS on the lock doubleword. The first half of the lock is an ID that uniquely identifies the owner. Language Environment must use the first four bytes of the thread ID of the calling thread as the lock owner ID, because this is guaranteed to be unique for the life of the owning thread, and is useful in providing problem determination support. The second half of the lock is a 3-byte count field of waiters for the mutex and a status byte. If the mutex is not in use and is not destroyed, the lock doubleword is all zeros. If the mutex is in use with no waiters, the lock owner ID field is nonzero, and the waiter count field is zero. If the mutex is in use with waiters, both fields are nonzero. The

kernel uses the information in the SMMX structure to provide cleanup and problem determination support for shared memory mutexes.

As an example, on a `pthread_mutex_lock` operation, Language Environment would attempt to set the first word of the lock to the first half (first four bytes) of the caller's thread ID, and set the waiter count field in the lock doubleword to its current value. If it can do this atomically with CDS, the mutex will be obtained without contention, and no call to BPX1SMC/BPX4SMC will be necessary. This allows a minimum of calls to the kernel.

The `pthread_mutex_init()` and `pthread_mutex_destroy` operations against a shared memory mutex are serialized by the BPX1SMC/BPX4SMC service and the use of the lock status byte. The lock status byte is updated atomically with CDS by BPX1SMC/BPX4SMC on a destroy operation, to indicate that the mutex has been destroyed. If the mutex is not in use, the CDS will succeed, and any further operations against the mutex will fail. BPX1SMC/BPX4SMC will also ensure that the same shared memory area is not initialized more than once.

- SMCV

The SMCV data area is created for shared memory condition variables. To represent the new larger type of shared mutex, the SMCV must have bits 0, 31, 62 and 63 on in the first 8 bytes of the structure. The characters SMCV (X'E2D4C3E5') are chosen as the ID (the first 4 bytes), so they meet the requirement for bits 0 and 31, and at the same time provide a representative eye-catcher for the structure. The second 4 bytes of the structure are a flag word, in which the low-order two flag bits in the word must always be initialized to being on, and must never be moved from their bit positions.

Serialization over this structure is provided mainly by the BPX1SMC/BPX4SMC service. This service must be called on all `pthread_cond` functions for a shared condition variable, unless there are no waiters for the condition variable on a `pthread_cond_signal`. Language Environment increments the waiter count on a `pthread_cond_wait` (or timed wait) before doing the call to BPX1SMC/BPX4SMC to wait and post, and decrements the count after waking up and receiving control back from the BPX1SMC/BPX4SMC service. The wait count can be incremented only if the `SmcvUnusable` flag is off in the lockword flag field. If this flag is on, the condition variable has been destroyed and is no longer usable. On a `pthread_cond_signal`, if the wait count is found to be zero, no call to the kernel is necessary.

## 12. CondTimedWait structure (SMCT)

The `CondTimedWait` structure is used for timed waits against shared condition variables, and contains the amount of time to wait in seconds and nanoseconds. The SMCT structure maps directly to the `timespec` structure that is currently supplied on the C `pthread_cond_timedwait()` function.

## 13. Shared memory remove

When a shared memory segment is removed with `shmctl()`, all condition variables and mutexes in the removed shared memory segment must be destroyed if the shared memory segment is to be cleaned up on the remove operation. This may involve waking up any waiters that are still using the mutex or condition variable that is being cleaned up. The actual cleanup for a shared memory segment may be delayed until the last attachor detaches. If the shared memory segment is not actually cleaned up on the remove operation, the shared condition variables and mutexes are still usable by the processes that are still attached to the shared memory segment.

## 14. Shared memory detach

When a shared memory segment is detached with `shmctl()`, the process loses access to the associated shared memory. If this is done before a process has stopped using all condition variables and mutexes in the detached shared memory segment, unpredictable results will occur in the application, possibly leading to hangs and loss of resources. Because a process is implicitly detached from its shared memory attachments when it terminates, it is recommended for most applications that they avoid doing any explicit `shmdt()` calls before terminating.

## 15. Process/thread termination

When a single thread ends, or when an entire process (possibly, many threads) ends while it is an owner of or a waiter for a shared memory mutex or condition variable, there may be waiters for the



mutex or associated condition variable that will get hung up if the application does not perform the necessary cleanup. To prevent hangs, kernel cleanup processing for all ending threads and processes is enhanced to perform cleanup for shared memory condition variables and mutexes, if necessary. If a thread ends (normally or abnormally) while it is a waiter for a shared mutex or condition variable, it is removed from the waiter list. If the thread ends abnormally, further cleanup is performed to ensure that the thread is not the owner of a shared mutex. This extra cleanup is not performed by the kernel for a normal termination, because Language Environment thread cleanup unlocks any mutexes that are held by a normally terminated thread.

In the case of an abnormally terminating thread, Language Environment thread cleanup may also unlock any held mutexes, if it gets a chance to run. If Language Environment cleanup does not unlock the held mutexes, kernel abnormal task termination will detect that the ending thread still owns a mutex. All of the waiters for a held mutex and the associated condition variable will be awoken. The mutex or condvar operation will then either abnormally terminate (X'EC6-xxxx8040' abend) or return an EINVAL, with the mutex and associated condition variable marked unusable. The object is marked unusable because the state of the data that the mutex or condition variable is serializing is uncertain, because of the abnormal ending of the thread that owned the resource. If BPX1SMC/BPX4SMC is abended while it is waiting for the unusable mutex or condvar, the EC6 abend is percolated to the caller of BPX1SMC/BPX4SMC. An unusable mutex or condition variable can only be destroyed.

16. No stub is provided in SYS1.CSSLIB to make the call successful. Instead, the method described in Appendix A must be used to invoke the service using the USS CSR table.

## Related services

- [“shmem\\_lock \(BPX1SLK, BPX4SLK\) — Shared memory lock service” on page 814](#)

## Characteristics and restrictions

None.

# shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment

---

## Function

The shmget function returns a shared memory ID that it either created or was allowed to access.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state; PSW key 2, 8, or 9
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1MGT):</b>	31-bit
<b>AMODE (BPX4MGT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1MGT, (Key,
               Shared_Memory_Size,
               Shared_Memory_Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MGT with the same parameters. The Shared\_Memory\_Size parameter is a doubleword.

## Parameters

### Key

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Identification for this shared memory segment. This is either a user defined value that serves as a lookup value to determine if the shared memory segment already exists, or the reserved value IPC\_PRIVATE. (See [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098. IPC\_PRIVATE is sometimes used when a process does not want to share a memory segment because it wants to privately control access to it by other processes.)

### Shared\_Memory\_Size

Supplied parameter

#### Type:

Integer

#### Length:

Fullword (doubleword)

A fullword (doubleword) field that contains the number of bytes of shared memory that are required. If IPC\_MEGA is specified, the value must be a multiple of megabytes, or the request is failed with EINVAL. If IPC\_GIGA is specified, the value must be a multiple of gigabytes, or the request is failed with EINVAL. If the caller is running in AMODE 64, the requested number of bytes will be rounded up to the nearest megabyte multiple.

### Shared\_Memory\_Flags

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Valid values for this field include any combination of the following (additional bits cause an EINVAL):

#### IPC\_CREAT

Creates a shared memory segment if the specified key does not already have an associated ID. IPC\_CREAT is ignored when IPC\_PRIVATE is specified.

#### IPC\_EXCL

Causes the shmget function to fail if the key specified has an associated ID. IPC\_EXCL is ignored when IPC\_CREAT is not specified or when IPC\_PRIVATE is specified.

#### IPC\_MEGA

Allocates shared storage in megabyte multiples. Use IPC\_MEGA to decrease ESQA storage utilization in support of shared memory segments.

**IPC\_GIGA**

Allocates shared storage in gigabyte multiples. Use IPC\_GIGA to decrease real storage utilization when running in AMODE 64.

**IPC\_BELOWBAR**

For AMODE 64 callers, IPC\_BELOWBAR forces the memory object to be allocated from below the 2-gigabyte address range. This allows 64-bit applications to share objects with 31-bit applications.

**IPC\_SHAREAS**

Enables sharing of the same storage area from multiple processes in the same address space.

- For AMODE 31 callers, this flag is only supported when the IPC\_MEGA flag is also specified; otherwise, this flag is ignored.
- For AMODE 64 callers, this flag is supported for all shared memory segments that are obtained above the 2G bar.

**S\_IRUSR**

Permits the process that owns the memory segment to read it.

**S\_IWUSR**

Permits the process that owns the memory segment to alter it.

**S\_IRGRP**

Permits the group that is associated with the memory segment to read it.

**S\_IWGRP**

Permits the group that is associated with the memory segment to alter it.

**S\_IROTH**

Permits others to read the memory segment.

**S\_IWOTH**

Permits others to alter the memory segment.

The values that begin with an "IPC\_" prefix are defined in BPXYIPCP. They are mapped onto S\_TYPE, which is in BPXYMODE. See [“BPXYIPCP — Map interprocess communication permissions”](#) on page 1098 and [“BPXYMODE — Map the mode constants ”](#) on page 1106.

The values that begin with an "S\_I" prefix are defined in BPXYMODE, and are a subset of the access permissions that apply to files.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmget service returns the shared memory identifier or, if it is unsuccessful, -1.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shmget service stores the return code. The shmget service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The shmget service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• The shared memory identifier does not exist for the Key parameter, and either the Shared_Memory-Size parameter is zero or it is greater than the system-imposed maximum. This system-imposed maximum is set with the IPCSHMMPAGES parameter in a BPXPRMxx parmlib member. You can use the <code>ipcs -x</code> shell command to view this value.</li> <li>• The shared memory identifier exists for the Key parameter, but the size of the segment that is associated with it is less than the Shared_Memory_Size parameter, and the Shared_Memory_Size parameter is not equal to 0.</li> <li>• The Shared_Memory_Flags include bits that are not supported by this function.</li> </ul> <p>The following reason codes can accompany the return code: JRShmBadSize and JRipcBadFlags.</p>
<b>EACCES</b>	<p>One of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• A shared memory identifier exists for the Key parameter, but the operation permission, as specified by the low-order 9-bits of the Shared_Memory_Flags parameter, is not granted (the "S_" items). The following reason code can accompany the return code: JRipcDenied.</li> <li>• The caller is running in PSW key 2 but has a TCB key other than 2. The following reason code can accompany the return code: JrKeyMismatch</li> </ul>
<b>EEXIST</b>	A shared memory identifier exists for the Key parameter, and both IPC_CREAT and IPC_EXCL were specified. The following reason code can accompany the return code: JRipcExists.
<b>ENOENT</b>	A shared memory identifier does not exist for the Key parameter, and IPC_CREAT was not specified. The following reason code can accompany the return code: JRipcNoExists.
<b>ENOMEM</b>	A shared memory identifier and associated shared memory segment are to be created, but the amount of system storage would exceed the system-imposed limit. The system limit is set with the IPCSHMSPAGES parameter in a BPXPRMxx parmlib member. You can use the <code>ipcs -y</code> shell command to view this value, which is displayed under SPAGES. The following reason code can accompany the return code: JRShmMaxSpages.
<b>ENOSPC</b>	A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allocated shared memory identifiers system-wide would be exceeded. This system limit is set with the IPCSHMNIDS parameter in a BPXPRMxx parmlib member. You can use the <code>ipcs -x</code> shell command to view this value. You can use the <code>ipcrm</code> shell command to remove unused shared memory identifiers. The following reason code can accompany the return code: JRipcMaxIDs.
<b>Reason_code</b>	
Returned parameter	
<b>Type:</b>	
Integer	
<b>Length:</b>	
Fullword	

The name of a fullword in which the shmget service stores the reason code. The shmget service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. When a shared memory segment has been created, subsequent shmget calls to find the existing shared memory segment must request a size that is less than or equal to the value that was specified when the shared memory segment was created.
2. As long as a task knows the shared memory segment ID, it may issue a shmat, shmctl or shmdt (shmget is not needed).
3. The shmget function returns the shared memory segment identifier that is associated with the Key parameter.
4. This function creates a data structure defined by SHMID\_DS if one of the following is true:
  - The Key parameter is equal to IPC\_PRIVATE.
  - The Key parameter does not already have a shared memory segment identifier associated with it, and IPC\_CREAT is set.

For the data structure, see [“BPXYSHM—Map interprocess communication shared memory segments”](#) on page 1147.

5. Upon creation, the data structure that is associated with the new shared memory segment identifier is initialized as follows:
  - IPC\_CUID and IPC\_UID are set to the effective user ID of the calling process.
  - IPC\_CGID and IPC\_GID are set to the effective group ID of the calling process.
  - The low-order 9-bits of IPC\_MODE are equal to the low-order 9-bits of the Shared\_Memory\_Flags parameter.
  - SHM\_OTIME is set to 0 and SHM\_CTIME is set to the current time.
  - The storage is initialized to nulls when the segment is created.
  - The storage is allocated in key 8.
6. The shared memory segment is removed from the system when shmctl RMID is processed, and when all users have detached (with the shmdt service) or terminated.
7. The first shmget request to define a shared memory segment determines whether the segment has the IPC\_MEGA/IPC\_GIGA attribute (on the IPC\_MEGA/IPC\_GIGA option of the Shared\_Memory\_Flags parameter). Subsequent shmget requests, which use existing shared memory segments, have no effect on the IPC\_MEGA attribute that is defined by that segment. In other words, the IPC\_MEGA/IPC\_GIGA option takes effect only for the first shmget request, and is ignored for all subsequent requests.
8. Shared memory segments created with the IPC\_MEGA/IPC\_GIGA attribute show this bit in the S\_MODE byte that is returned with the w\_getipc request.
9. Above the bar, shared memory cannot be used in subspace mode.
10. The user address space storage for a shared memory segment is normally obtained in storage key 8, except under the following special circumstances:
  - The caller of BPX1MGT that initially creates a shared memory segment is running in PSW key 2 or 9 and either of the following is true:
    - The caller is running in AMODE 64 and the shared memory segment is neither of type IPC\_MEGA nor type IPC\_BELOWBAR.
    - The caller is running in AMODE 31 and the shared memory segment is of type IPC\_MEGA.

Under these circumstances, the user address space storage is obtained in the key of the caller (either key 2 or key 9). Any subsequent use of the segment from any address space will cause the user address space storage to be obtained in the same storage key in which the segment was initially

created. This is true regardless of the PSW key in which the caller is running at the time of any subsequent attach request.

## Related services

- [“w\\_getipc \(BPX1GET, BPX4GET\) — Query interprocess communications” on page 991](#)
- [“shmat \(BPX1MAT, BPX4MAT\) — Attach to a shared memory segment ” on page 804](#)
- [“shmctl \(BPX1MCT, BPX4MCT\) — Perform shared memory control operations ” on page 809](#)
- [“shmdt \(BPX1MDT, BPX4MDT\) — Detach a shared memory segment” on page 812](#)

## Characteristics and restrictions

- There is a maximum number of shared memory segments allowed in the system.
- There is a system-imposed limit on the maximum segment size that is defined in the BPXPRMxx parmlib member.
- The caller of the shmget service is restricted by ownership and read and read-write permissions that are defined by shmget and shmctl IPC\_SET.

## Examples

See [“BPX1MGT \(shmget\) example” on page 1260](#).

# shutdown (BPX1SHT, BPX4SHT) — Shut down all or part of a duplex socket connection

---

## Function

The shutdown callable service shuts down all or part of a duplex socket connection.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SHT):</b>	31-bit task or SRB mode
<b>AMODE (BPX4SHT):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SHT,(Socket_descriptor,
               How,
               Return_value,
```

Return\_code,  
Reason\_code)

AMODE 64 callers use BPX4SHT with the same parameters.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the shutdown is to be done.

### How

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the condition of the shutdown:

- 0 ends communication from Socket (Read).
- 1 ends communication to Socket (Write).
- 2 ends communication both to and from Socket.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the shutdown service returns one of the following:

- 0 if the request is successful.
- -1 if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the shutdown service stores the return code. The shutdown service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The shutdown service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	An incorrect file descriptor was supplied. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>EINVAL</b>	The How parameter is incorrect. It is not 0, 1, or 2. The following reason code can accompany the return code: JRBadEntryCode.

Return_code	Explanation
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the shutdown service stores the reason code. The shutdown service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. See [Chapter 3, “Callable services for SRB mode routines,”](#) on page 1041 for more information about programming considerations for SRB mode.
2. A shutdown for read means that future write operations from the other end of this socket are rejected. Any data that was already written before the shutdown occurred are available for the application that issued the shutdown to read. The data is read until a read is done that returns zero bytes, indicating that there is no more data for that socket.
3. A shutdown for write means that any future writes by the application that issued the shutdown request are rejected.
4. Regardless of the How option specified, reads are not rejected.

**Characteristics and restrictions**

None.

**Examples**See [“BPX1SHT \(shutdown\) example”](#) on page 1288.

## **sigaction (BPX1SIA, BPX4SIA) – Examine or change a signal action**

**Function**

The sigaction callable service examines, changes, or both examines and changes the action that is associated with a specific signal for all threads in the process.

**Note:** The signal handlers, a set of additional signals to be masked, and flags that are specified by the sigaction service are shared by all threads within a process.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN



<b>Operation</b>	<b>Environment</b>
<b>AMODE (BPX1SIA):</b>	31-bit
<b>AMODE (BPX4SIA):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1SIA, (Signal,
               New_sa_handler_address,
               New_sa_mask,
               New_sa_flags,
               Old_sa_handler_address,
               Old_sa_mask,
               Old_sa_flags,
               User_data,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SIA with the same parameters. All parameter addresses and addresses in parameter structures are doublewords. The User\_data parameter is a doubleword field.

Parameters

Signal

Supplied parameter

Type:

Integer

Length:

Fullword

The name of a fullword that contains the number of the signal to examine, set, or both set and examine the action for.

New\_sa\_handler\_address

Supplied parameter

Type:

Address

Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains either zero or the address of a fullword that contains the new signal action.

- If it contains zero, no new action is set for this signal.
- If it is not zero, set the signal action using the options that are described in this topic and in the BPXYSIGH macro. See [“BPXYSIGH — Signal constants” on page 1148](#).

Constant

Description

**SIG\_DFL#**

Take the default action for this signal.

**SIG\_IGN#**

Ignore this signal.

Constant	Description
<b>Address</b>	Address of the signal catcher function.
<b>New_sa_mask</b> Supplied parameter	
<b>Type:</b> Structure	
<b>Length:</b> 8 bytes	
The name of an 8-byte area that contains a 64-bit mask of signals that are to be blocked during execution of the signal-catching function. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. Bits that are set to 1 represent signals that are blocked.	
You must always provide this field, even though New_sa_mask is not used when the New_sa_handler_address parameter contains a 0.	
<b>New_sa_flags</b> Supplied parameter	
<b>Type:</b> Structure	
<b>Length:</b> Fullword	
The name of the fullword that contains the value of the signal action flags.	
You must always provide this field, even though New_sa_flags is not used when the New_sa_handler_address parameter contains a 0.	
New_sa_flags can be set to the following constants defined in the BPXYSIGH macro:	
Constant	Description
<b>SA_FLAGS_DFT#</b>	None of the following functions.
<b>SA_NOCLDSTOP#</b>	Do not generate SIGCHLD signals to the calling process when its children stop or are continued. (This is used only when Signal is set to SIGCHLD).
<b>SA_OLD_STYLE#</b>	The PPSDOLDSTYLE flag is set. This is provided for the C compiler runtime library to implement old-style signal callable service functions. The C compiler runtime library's signal interface routine is responsible for checking PPSDOLDSTYLE and setting sigaction to default action where needed.
<b>SA_ONSTACK#</b>	The PPSDONSTACK flag is set. This is provided for the caller to implement alternate stack signal delivery processing.
<b>SA_RESETHAND#</b>	The PPSDRESETHAND flag is set. This is provided for the caller to reset the signal action to SIG_DFL# on entry to the signal catcher.
<b>SA_RESTART#</b>	The PPSDRESTART flag is set. This is provided for the caller to implement restart for functions that normally would receive an EINTR if a signal is delivered.
<b>SA_SIGINFO#</b>	The PPSDSIGINF flag is set. This is provided for the caller to provide additional information to the signal catcher, namely additional signal information and user context information.

Constant	Description
<b>SA_NOCLDWAIT#</b>	Do not create zombie processes when children of the calling process exit (used only when Signal is set to SIGCHLD).
<b>SA_NODEFER</b>	The PPSDNODEFER flag is set. This is provided for the caller to not automatically block the Signal when the signal catcher is invoked.
<b>SA_IGNORE</b>	The value of the new_sa_handler is saved and returned on subsequent calls, but the signal is ignored.
<b>Old_sa_handler_address</b>	
Parameter supplied and returned	
<b>Type:</b>	
Address	
<b>Length:</b>	
Fullword (doubleword)	
The name of a fullword (doubleword) that contains either zero or the address of a fullword in which the system returns the old (current) signal action. If Old_sa_handler_address is specified as 0, the old signal action, Old_sa_mask, and Old_sa_flags, are not returned.	
<b>Old_sa_mask</b>	
Returned parameter	
<b>Type:</b>	
Structure	
<b>Length:</b>	
8 bytes	
The name of an 8-byte area where the old (current) value of the 64-bit mask of signals blocked during execution of the signal-catching function is returned. Bits that are set to 1 represent signals that are blocked.	
You must provide this parameter, although Old_sa_mask is not returned when Old_sa_handler_address contains 0.	
<b>Old_sa_flags</b>	
Returned parameter	
<b>Type:</b>	
Structure	
<b>Length:</b>	
Fullword	
The name of the fullword in which the old (current) signal action flags are returned.	
You must always provide this field, even though Old_sa_flags is not returned when Old_sa_handler_address contains 0.	
<b>User_data</b>	
Supplied parameter	
<b>Type:</b>	
Character	
<b>Length:</b>	
Fullword (doubleword)	
The name of a fullword (doubleword) that contains 4 bytes of user-supplied data that is passed to the signal interface routine when the signal is delivered.	

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigaction service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigaction service stores the return code. The sigaction service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sigaction service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The specified signal value is incorrect or is an unsupported signal number; an attempt was made to catch a signal that cannot be caught; or an attempt was made to ignore a signal that cannot be ignored. The following reason codes can accompany the return code: JRInvalidSignal and JRInvalidSigact.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigaction service stores the reason code. The sigaction service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If New\_sa\_handler\_address value is set to the action SIG\_DFL for a signal that cannot be caught or ignored, the sigaction request is ignored and Return\_value is set to 0.
2. Setting a signal action to ignore for a signal that is pending causes the pending signal to be discarded.
3. Setting signal action SIG\_IGN or catch for signals SIGSTOP, SIGTSTP, SIGTCONT, or SIGKILL is not allowed.
4. Setting signal action SIG\_IGN for SIGIO is not allowed.
5. The SA\_NOCLDWAIT flag should not be used with the waitpid (BPX1WAT, BPX4WAT) WNOHANG flag, or with the SIGSTOP or SIGCONT signals. Because the SA\_NOCLDWAIT flag indicates that child processes of the calling process do not become zombies, these child processes do not report their status to the calling process when they end. Thus, if the calling process subsequently issues a waitpid call, it suspends until all of its child processes terminate, and then receives an ECHILD error return. This is expected behavior when SA\_NOCLDWAIT is set. However, using SIGSTOP or SIGCONT signals with a child process could cause stop status to be reported to the calling process if it issues a subsequent waitpid call. Likewise, the use of the WNOHANG flag on a subsequent waitpid would

result in an immediate return, instead of the process suspending until all child processes terminate. For these reasons, care should be taken not to mix these incompatible functions.

6. Setting signal action SIG\_IGN for SIGCHLD has the same effect as setting the SA\_NOCLDWAIT flag.
7. The SA\_NOCLDSTOP and SA\_NOCLDWAIT flags, despite having similar names, result in different types of actions. SA\_NOCLDSTOP results in SIGCHLD signals not being sent when child processes stop or are continued. Setting SA\_NOCLDWAIT results in child processes not becoming zombies or reporting their exit status, but SIGCHLD signals are still sent when child processes end.
8. The user data is delivered on a per-signal basis for the specific signal that is specified on this invocation. This field must be respecified if user data is desired for the next signal. This user data is set even if the action is SIG\_DFT or SIG\_IGN.
9. Although the user can be in supervisor state and any PSW key when this service is used, the kernel does not deliver signals to the signal interface routine until the task is running with a PSW key equal to the PSW key when the first callable service was entered and the process was created. See [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#).
10. The sigaction caller's thread must be registered for signals. You can register the thread by calling mvssigsetup, or, after signals are set up, by creating the thread with pthread\_create. If the thread is not registered for signals, the sigaction service fails with a return code of EINVAL and a reason code of JRNotSigSetup. See [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#).
11. Constants that are used for this callable service are defined in the BPXYSIGH macro. See [“BPXYSIGH — Signal constants” on page 1148](#).

## Related services

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered” on page 851](#)

## Characteristics and restrictions

In a multithreaded process, the new signal action that is set by the sigaction service changes the signal action for all threads in the process.

## Examples

See [“BPX1SIA \(sigaction\) example” on page 1288](#).

# **\_\_sigactionset (BPX1SA2, BPX4SA2) — Examine or change a set of signal actions**

---

## Function

The \_\_sigactionset callable service examines, changes, or both examines and changes the actions that are associated with a set of signals.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task

<b>Operation</b>	<b>Environment</b>
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SA2):</b>	31-bit
<b>AMODE (BPX4SA2):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SA2, (New_count,
               New_structure,
               Old_count,
               Old_structure,
               SsetOption_flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SA2 with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### New\_count

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the number of array elements in New\_structure. New\_count must be in the range from 0 to 64. If New\_structure is not provided, specify a count of 0.

### New\_structure

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address that points to the beginning of the new structure. The new structure contains the layout of the desired parameters for sigaction: New\_sa\_handler\_address, New\_sa\_flags, New\_sa\_mask, UserData, and ConsolMask. ConsolMask is a bit mask that defines all the signals that are to have the same action. New\_sa\_handler\_address, New\_sa\_flags, New\_sa\_mask, and UserData are mapped by the BPXYSSSET macro. See [“BPXYSSSET — Map the sigaction set”](#) on page 1160.

### Old\_count

Supplied and returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword that contains the number of array elements that are allowed within Old\_structure on input. On output, Old\_count contains the number of array elements that have been stored. If Old\_count is too small to hold the number of array elements that are needed, return code ENOMEM is returned. When ENOMEM is returned, Old\_count contains the number of array elements that are required to contain the current signal action state.

Old\_count must be in the range from 0 to 64.

If Old\_structure is not provided, specify a count of 0.

You may not pass a constant in Old\_count. If a constant is passed, an EFAULT is generated when an attempt is made to store back the result on exit.

**Old\_structure**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address that points to the beginning of the old structure. On output from the call to \_\_sigactionset, Old\_structure contains the number of signal actions specified in Old\_count.

**SsetOption\_flags**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of the area in which the option flags are set. A leftmost bit (Sset\_IgInvalid) set to 1 indicates signals that are not valid; signals that are not valid are to be ignored. Possible SsetOption\_Flags: Sset\_IgInvalid = X'80000000', which indicates that invalid signals and sigactions are to be ignored.

In the following example, Sset\_IgInvalid is set to 1 and New\_count is passed in as 3. New\_structure has been given an address that points to the storage area that contains the five fields shown: ConsolidatedMask, New\_sa\_flags, New\_sa\_mask, and New\_user\_data.

NewStruct						
		ConsolidatedMask (64 Bits)	New SaFlags (4 Bytes)	New SaHandler (4 Bytes)	New SaMask (64 Bits)	New UserData (4 Bytes)
NewCount		X'0FFFFFFFFFFFFFFFFF'	0	0	0	0
1	→	X'3000000000000000	0	1	0	0
2	→	X'C000000000000000'	0	0480E000	'01..10'B	0
3	→					

**Note:**

1. New\_count can range from 1 to the maximum number of signals.
2. The signal handlers (a set of additional signals to be masked), option flags, and user data that is specified by the \_\_sigactionset service, are shared by all threads within a process.
3. In the example shown:
  - The first set defines the action for signals 5–64 to their default state. Because some of these signals are unsupported, the setting of SsetOption\_flags (Sset\_IgInvalid) tells \_\_sigactionset to ignore unsupported signals.
  - The second set tells \_\_sigactionset to ignore signals 3 (SIGABRT#) and 4 (SIGILL#).
  - The third set defines a signal catcher with a mask for signals 1 (SIGHUP#) and 2 (SIGINT#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_sigactionset service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_sigactionset service stores the return code. The \_\_sigactionset service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_sigactionset service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EINVAL**

The specified signal is incorrect or is an unsupported signal number; an attempt was made to catch or ignore a signal that cannot be caught or ignored; or the specified signal value was not within the range from 0 to 64. The following reason codes can accompany the return code: JRInvalidSignal, JRInvalidSigact, and JRInvalidRange.

The following are examples of incorrect scenarios:

- Sset\_IgInvalid is set to 0, and any bit position ranging from 33 to 64 is on. JRInvalidSignal is returned.
- Sset\_IgInvalid is set to 0, and signal 7 (SIGSTOP#), signal 34 (SIGTSTP#), signal 35 (SIGTCONT#), or signal 9 (SIGKILL#) is on. JRInvalidSigact is returned.

**ENOMEM**

There is not enough memory available to hold the number of array elements required to contain the current signal action state. The following reason codes can accompany the return code: JRSsetTooSmall.

**Reason\_code**

Returned parameter

**Type:**

Integer



**Length:**

Fullword

The name of a fullword in which the `__sigactionset` service stores the reason code. The `__sigactionset` service returns `Reason_code` only if `Return_value` is `-1`. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. For full details about the `sigaction()` parameters, see [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action”](#) on page 834.
2. In a multithreaded process, the new signal action that is set by the `__sigactionset` service changes the signal action for all threads in the process.
3. If multiple masks have a bit set on for the same signal, the one that is set is the last one.
4. If the caller of `__sigactionset` does not specify `Sset_IgInvalid` within `SsetOption_flags`, a return code of `EINVAL` is returned for all signals and sigactions that are not valid. You can bypass this error by setting `Sset_IgInvalid` to 1.
5. If `New_count` is zero (indicating a query of old signal actions), no changes are made to the signal actions.
6. If `Old_count` is zero, the `__sigactionset` service does not return anything in `Old_structure`.

**Related services**

- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action”](#) on page 834
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program”](#) on page 163
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process”](#) on page 360
- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals”](#) on page 487
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask”](#) on page 845
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered”](#) on page 851

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1SA2 \(\\_\\_sigactionset\) example”](#) on page 1284.

## **sigpending (BPX1SIP, BPX4SIP) — Examine pending signals**

---

**Function**

The `sigpending` service returns the union of the set of signals that are pending on the thread and the set of signals that are pending on the process.

Pending signals at the process level are moved to the thread that called the `sigpending` service.

**Requirements****Operation****Environment****Authorization:**

Supervisor state or problem state, any PSW key

**Dispatchable unit mode:**

Task

Operation	Environment
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SIP):</b>	31-bit
<b>AMODE (BPX4SIP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SIP, (Signal_pending_mask,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SIP with the same parameters.

## Parameters

### Signal\_pending\_mask

Returned parameter

#### Type:

Structure

#### Length:

8 bytes

The name of an 8-byte area to which the sigpending service returns a 64-bit signal pending mask. Bits that are set on represent signals that are pending and blocked. Each bit that is set to on represents a signal that is currently pending at the process level or the thread-level and is blocked by the current thread's signal mask. The leftmost bit represents signal 1, and the rightmost bit represents signal 64.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigpending service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigpending service stores the return code. The sigpending service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigpending service stores the reason code. The sigpending service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ”](#) on page 845

**Characteristics and restrictions**

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,”](#) on page 1413.

**Examples**

See [“BPX1SIP \(sigpending\) example”](#) on page 1289.

## sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask

---

**Function**

The sigprocmask callable service examines, changes, or both examines and changes the calling thread's signal mask.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SPM):</b>	31-bit
<b>AMODE (BPX4SPM):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SPM, (How,
               New_signal_mask,
               Old_signal_mask,
               Return_value,
```

```
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4SPM with the same parameters.

## Parameters

### How

Supplied parameter

#### Type:

Structure

#### Length:

Fullword

The name of a fullword that contains a numeric value that identifies the action that is to be taken on the thread's signal mask. The following constants, which are defined in BPXYSIGH, define the actions to be taken: See [“BPXYSIGH — Signal constants”](#) on page 1148.

#### Constant

#### Description

##### **SIG\_BLOCK#**

Add the signals in New\_signal\_mask to those to be blocked for this thread.

##### **SIG\_UNBLOCK#**

Delete the signals in New\_signal\_mask from those blocked for this thread.

##### **SIG\_SETMASK#**

Replace the thread's signal mask with New\_signal\_mask.

### New\_signal\_mask

Supplied parameter

#### Type:

Address

#### Length:

Fullword

The name of a fullword that contains either 0 or the address of an 8-byte area that contains the 64-bit signal mask. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. The New\_signal\_mask parameter is applied to the thread's signal mask as specified by the How parameter. Mask bits that are set on represent signals that are blocked. If zero, the signal mask is not changed, and the How parameter is ignored.

### Old\_signal\_mask

Parameter supplied and returned

#### Type:

Address

#### Length:

Fullword

The name of a fullword that contains either 0 or the address of an 8-byte signal-mask return area. The service stores in this area the signal mask that was in effect, showing the signals that were blocked. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. Mask bits set on represent signals that are blocked. A zero indicates that no signal mask was returned.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigprocmask service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigprocmask service stores the return code. The sigprocmask service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sigprocmask service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EFAULT</b>	The specified address for New_signal_mask or Old_signal_mask was incorrect.
<b>EINVAL</b>	The value of the How parameter is not one of the allowable values.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigprocmask service stores the reason code. The sigprocmask service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The sigprocmask service examines, changes, or both examines and changes the signal mask for the calling thread. This mask is called the thread's signal mask. If there are any pending unblocked signals, either at the process level or at the current thread's level, after changing the signal mask, at least one of the signals is delivered to the thread before the sigprocmask service returns.
2. In a multithreaded process, the sigprocmask service is used to control to which thread in the process a signal that is generated by the kill service is delivered. For example, if two threads in a process have SIGUSR1 signals blocked and one thread does not, the SIGUSR1 signal that is generated by the kill service from another process is delivered to the thread that does not have the signal blocked.
3. You cannot block the SIGKILL, SIGSTOP, SIGTHSTOP, and SIGTHCONT signals. If you call the sigprocmask service with a request that would block those signals, that part of your request is ignored and no error is indicated.
4. A request to block signals that are not supported is accepted, and a return value of zero is returned.
5. All pending unblocked signals are moved from the process level to the current thread.
6. See [Appendix I, "Optimizing performance using process- and thread-level information,"](#) on page 1431.

## Related services

- ["kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process"](#) on page 360
- ["mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals"](#) on page 487
- ["sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action"](#) on page 834

- “sigpending (BPX1SIP, BPX4SIP) — Examine pending signals ” on page 843
- “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 851

**Characteristics and restrictions**

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,”](#) on page 1413.

**Examples**

See “BPX1SPM (sigprocmask) example” on page 1293.

## sigqueue (BPX1SGQ, BPX4SGQ) — Queue a signal to a process

---

**Function**

The sigqueue callable service queues a signal to a process, a process group, or all processes in the system to which the caller has permission to queue a signal.



**CAUTION:** Note that when a caller with appropriate privileges (see “Authorization” on page 8) specifies a process ID equal to -1, the signal will normally be queued to all processes in the system, excluding the INIT process (process ID 1). If the signal action is to terminate the process, all processes will terminate. This may not be the desired action, considering that some processes may be necessary for the continued operation of the system.

**Requirements****Operation****Authorization:****Dispatchable unit mode:****Cross memory mode:****AMODE (BPX1SGQ):****AMODE (BPX4SGQ):****ASC mode:****Interrupt status:****Locks:****Control parameters:****Environment**

Problem program or supervisor state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary address space control (ASC) mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SGQ,(Process_ID,
               Signal,
               Signal_Value,
               Signal_Options,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SGQ with the same parameters. Signal\_Value is a doubleword field.

## Parameters

### Process\_ID

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword whose value specifies the process or processes to which the caller wants to queue a signal:

- If Process\_ID is greater than 0, it is assumed to be a process ID. The signal is queued to the process with that process ID.
- If Process\_ID is equal to 0, the signal is queued to all processes that have a process group ID equal to that of the caller, and for which the caller has permission to queue the signal.
- If Process\_ID is -1, the signal is queued to all processes for which the caller has permission to queue the signal.
- If Process\_ID is less than -1, its absolute value is assumed to be a process group ID. The signal is queued to all processes that have a process group ID equal to this absolute value, and for which the caller has permission to queue a signal.

Note the restrictions in [“Characteristics and restrictions”](#) on page 851.

### Signal

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the signal number that is to be queued to the processes indicated by the Process\_ID parameter. The signal number must be defined in the BPXYSIGH macro, or it must be 0. See [“BPXYSIGH — Signal constants”](#) on page 1148.

If the signal is 0, error checking takes place, but no signal is queued. Use a signal value of 0 to verify that the Process\_ID parameter is correct before actually queuing a signal. This method does not verify permission to queue the signal to the specified Process\_ID.

### Signal\_Value

Supplied parameter

#### Type:

Integer

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains data to be delivered with the signal.

### Signal\_Options

Supplied parameter

#### Type:

Structure

#### Length:

Fullword

The name of a fullword that contains the binary flags that describe how the signal is to be handled by the system and the user-supplied signal interface routine (SIR). This byte of user information is passed to the SIR in a data structure mapped by the PBXYPPSD macro. See [“PBXYPPSD — Map signal delivery data”](#) on page 1124.

Signal options are mapped as follows:

**First 2 bytes**

User-defined bytes to be delivered with the signal to the SIR in the signal information control block. These bytes are mapped by PPSDKILDATA.

**Last 2 bytes**

Flag bits, mapped by PPSDKILOPTS, that are defined as follows:

- First bit - signal to bypass Ptrace processing
- Second bit - reserved
- Third bit - the signal code specified in the first 2 bytes is set by the application
- Remaining bits - reserved

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigqueue service returns 0 if it has permission to queue the specified signal to any of the processes specified by the Process\_ID parameter. A return value of 0 means that a signal was queued (or could have been queued, if the signal value was 0) to at least one of the specified processes.

If a signal is not queued, the return value is -1.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigqueue service stores the return code. The sigqueue service stores a return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sigqueue service may return one of the following values in the Return\_code parameter:

<b>Return code</b>	<b>Explanation</b>
<b>EAGAIN</b>	The caller has reached the maximum number of queued signals (MAXQUEUEDSIGS) allowed in a process.
<b>EINVAL</b>	The value specified in the Signal parameter is incorrect, or not the number of a supported signal.
<b>EMVSSAF2ERR</b>	The SAF ck_process_owner (IRRSK000) callable service returned with an unexpected error.
<b>EPERM</b>	The caller does not have permission to queue the signal to any process specified in the Process_ID parameter.
<b>ESRCH</b>	No processes or process groups corresponding to the value specified in the Process_ID parameter were found.

**Reason\_code**

Returned parameter

**Type:**

Integer



**Length:**

Fullword

The name of a fullword in which the sigqueue service stores the reason code. The sigqueue service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR, the reason code contains the security product return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the security product Check Privilege service return and reason code values, see the following table:

Security product return code	Security product reason code	Explanation
8	4	The caller is not the owner of the target process.
8	12	There was an internal error during security product processing.

**Related services**

- [“getpid \(BPX1GPI, BPX4GPI\) — Get the process ID” on page 298](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ” on page 777](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“setsid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ” on page 793](#)

**Characteristics and restrictions**

1. A caller can queue a signal if the real or effective user ID of the caller is the same as the real or saved set user ID of the intended recipient. A caller can also queue signals if it has appropriate privileges (see [“Authorization” on page 8](#)).
2. Regardless of its user ID, a caller can always queue a SIGCONT signal to a process that is a member of the same session.
3. A caller can queue a signal to itself. If the signal is not blocked, at least one pending unblocked signal is delivered to the caller before the service returns control. Provided that no other unblocked signals are pending, the signal that is delivered is the signal that is queued. See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413](#) for more information.
4. Note the caution at the beginning of this callable service description.

**Examples**

See [“BPX1SGQ \(sigqueue\) example” on page 1287](#).

## sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered

---

**Function**

The sigsuspend callable service replaces a thread's current signal mask with a new signal mask. It then suspends the caller's thread until delivery of a signal whose action is either to process a signal-catching service or to end the thread.

## Requirements

### Operation

#### Authorization:

### Environment

Problem Program or Supervisor State, PSW key when the process was created (not PSW key 0)

#### Dispatchable unit mode:

Task

#### Cross memory mode:

PASN = HASN

#### AMODE (BPX1SSU):

31-bit

#### AMODE (BPX4SSU):

64-bit

#### ASC mode:

Primary mode

#### Interrupt status:

Enabled for interrupts

#### Locks:

Unlocked

#### Control parameters:

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SSU, (Signal_mask,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SSU with the same parameter.

## Parameters

### Signal\_mask

Supplied parameter

#### Type:

Structure

#### Length:

8 bytes

The name of an 8-byte area that contains a 64-bit signal mask that is set before waiting for a signal, and during the execution of any signal catcher. The leftmost bit represent signals 1 and the rightmost bit represents signal 64. Bits that are set to 1 represent signals that are blocked.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigsuspend service returns a -1 if it returns to its caller.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigsuspend service stores the return code. The sigsuspend service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sigsuspend service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EINTR	A signal was received and handled successfully.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sigsuspend service stores the reason code. The sigsuspend service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### Usage notes

1. The caller's thread starts running again when it receives one of the signals that are not blocked by the mask that is set by this call, or when a system failure occurs that sets Return\_code to some value other than EINTR.
2. The signal mask represents a set of signals that are to be blocked. Blocked signals do not "wake up" the suspended service. The signals SIGSTOP, SIGTHSTOP, SIGTHCONT, and SIGKILL cannot be blocked or ignored; they are delivered to the program no matter what the signal mask specifies.
3. If the signal action is to end the thread, the sigsuspend service does not return.
4. If the signal interruption is to give control to the signal interface routine (SIR), which is defined by the mvssigsetup service, the SIR is given control with the following PPSD fields:

#### PPSDSAMASK

Set to the New\_sa\_mask value, which is set by the sigaction service, for the signal number that caused the interruption.

#### PPSDCURRENTMASK

The signal mask value that existed before the sigsuspend service was called.

To be XPG4 compliant, this is the signal mask that is installed when a signal catcher performs a normal return.

#### PPSDCATCHERMASK

The signal mask that is specified on the sigsuspend service.

To be XPG4 compliant, the signal mask that is installed before calling a signal catcher is calculated by taking the union of PPSCATCHERTMASK, PPSDSAMASK, and the signal that caused the interrupt.

5. The signal interface routine (SIR) that is defined by the mvssigsetup service is given control only when the PSW key of the sigsuspend caller is equal to the signal catcher key of the process. The signal catcher key is set to the PSW key of the caller of the first z/OS UNIX callable service that created the process.
6. If the caller has a PSW key that is different from the signal catcher key, or has a PSW key of zero, the sigsuspend service returns with a return code of EMVSERR and reason code of JRPSWKeyNotValid.
7. All pending unblocked signals are moved from the process level to the current thread.

## Related services

- “[pause \(BPX1PAS, BPX4PAS\)](#) — Suspend a process pending a signal” on page 539
- “[sigaction \(BPX1SIA, BPX4SIA\)](#) — Examine or change a signal action ” on page 834
- “[sigpending \(BPX1SIP, BPX4SIP\)](#) — Examine pending signals ” on page 843
- “[sigprocmask \(BPX1SPM, BPX4SPM\)](#) — Examine or change a process's signal mask ” on page 845

## Characteristics and restrictions

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,”](#) on page 1413.

## Examples

See “[BPX1SSU \(sigsuspend\) example](#)” on page 1297.

# sigtimedwait (BPX1STW, BPX4STW) – Wait for a signal with a specified timeout

---

## Function

The sigtimedwait callable service suspends the invoking thread until either the specified timeout expires, or a signal specified in the signal set becomes pending, at either the process or the invoking thread. If a signal that is specified in the signal set is sent to the invoker of sigtimedwait, the value of that signal is returned to the invoker and the sigtimedwait service ends.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1STW):</b>	31-bit
<b>AMODE (BPX4STW):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1STW,(Signal_mask,
              Siginfo_ptr,
              Siginfo_len,
              Seconds,
              Nanoseconds,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STW with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Signal\_mask

Supplied parameter

#### Type:

Structure

#### Length:

8 bytes

The name of an 8-byte field area that contains a 64-bit signal mask that contains the set of signals that this task is to wait on. The leftmost bit represents signal 1, and the rightmost bit represents signal 64. Bits that are set to 1 represent signals that are waited on.

### Siginfo\_ptr

Supplied parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of user-supplied storage that is mapped by the BPXYSINF macro (see [“BPXYSINF — Map SIGINFO\\_T structure”](#) on page 1150). If this address is nonzero, the sigtimedwait service uses this area to place additional signal information when a signal number is returned in Return\_value. If this address is zero, or if an error is returned, no additional information is returned in this area.

### Siginfo\_len

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the user-supplied storage that is mapped by the BPXYSINF macro. If the address of Siginfo\_ptr is zero, this parameter is ignored.

### Seconds

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains an unsigned integer representing the maximum number of seconds that the calling program is willing to wait for one of the specified signals to become pending.

#### Note:

1. Seconds can be any value greater than or equal to 0 and less than or equal to 4 294 967 295. The value specified for Seconds must be an unsigned integer.
2. The values in the Seconds and Nanoseconds parameters are combined to determine the timeout value. A combined value of zero indicates that the sigtimedwait service does not wait at all. A value of SIG#NO\_TIMEOUT (see [“BPXYSIGH — Signal constants”](#) on page 1148) indicates that no timeout value is set.

**Nanoseconds**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains an unsigned integer representing the number of nanoseconds to be added to the value specified by the Seconds parameter. Nanoseconds can be any value greater than or equal to 0 and less than or equal to 1 000 000 000.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigtimedwait service returns the signal if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigtimedwait service stores the return code. The sigtimedwait service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sigtimedwait service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	One or more of the parameters that were passed to this service are in error. The following reason codes unique to the sigtimedwait service can accompany the return code: JRNanoSecondsTooBig, JRInvalidSignal.
<b>EAGAIN</b>	The service timed out before any of the specified signals became pending on the invoking thread.
<b>EINTR</b>	The service received a signal that was not specified in the input signal mask.

**Reason\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigtimedwait service stores the reason code. The sigtimedwait service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The sigtimedwait service behaves in the same way as the sigwait service, except that with the sigtimedwait service you can specify a timeout value.
2. If a nonzero address is specified for the siginfo\_ptr parameter, the sigtimedwait service also returns si\_signo, si\_code, and si\_value, as mapped by BPXYSINF.
3. A timeout value of zero (Seconds + Nanoseconds) means that the sigtimedwait service does not wait at all. It checks for pending signals, and if no signal is found, it returns with an error of EAGAIN. If a signal is found, the service returns with the signal number of the pending signal.
4. A passed timeout value of SIG#NO\_TIMEOUT (see “BPXYSIGH — Signal constants” on page 1148) indicates that no timeout value is set. The sigtimedwait service waits until a signal becomes pending.

## Related services

- “sigwait (BPX1SWT, BPX4SWT) — Wait for a signal ” on page 857
- “kill (BPX1KIL, BPX4KIL) — Send a signal to a process ” on page 360
- “sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask ” on page 845

## Characteristics and restrictions

None.

## Examples

See “BPX1STW (sigtimedwait) example” on page 1299.

# sigwait (BPX1SWT, BPX4SWT) — Wait for a signal

---

## Function

The sigwait callable service waits for an asynchronous signal. If a signal that is specified in the signal set is sent to the invoker of sigwait, the value of that signal is returned to the invoker and the sigwait service ends.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SWT):</b>	31-bit
<b>AMODE (BPX4SWT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SWT,(Signal_mask,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4SWT with the same parameters.

**Parameters**

**Signal\_mask**

Supplied parameter

**Type:**

Structure

**Length:**

8 bytes

The name of an 8-byte field area that contains a 64-bit signal mask that contains the set of signals that this task is to wait on. The leftmost bit represents signal 1, and the rightmost bit represents signal 64. Bits that are set to 1 represent signals that are waited on.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigwait service returns the signal if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigwait service stores the return code. The sigwait service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sigwait service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EINVAL	The Signal_mask argument contained a signal that represents an incorrect signal number. The following reason code can accompany this return code: JRInvalidSignal.

**Reason\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sigwait service stores the reason code. The sigwait service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



## Usage notes

1. If any signals that are specified in `Signal_mask` are pending upon invocation of the `sigwait` service, one of those signals has its value returned to the invoker, and that signal is cleared from the set of pending signals.
2. If there are no pending signals that were specified in `Signal_mask`, the `sigwait` service waits until a signal that is specified in `Signal_mask` is generated. A signal mask of zero causes the caller to wait until the task or process is terminated.
3. If `sigwait` is invoked for a `SIGKILL`, `SIGSTOP`, or `SIGTHSTOP` signal, and a `SIGKILL`, `SIGSTOP`, or `SIGTHSTOP` signal arrives, the value of the signal is not returned to the invoker. Rather, the `SIGKILL`, `SIGSTOP`, or `SIGTHSTOP` action occurs.
4. The current `sigaction` ( [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action” on page 834](#)) that is associated with a signal that is returned is not performed. This action also remains unchanged by the use of the `sigwait` service.
5. If there are multiple threads in a process that have issued a `sigwait` for the same signal, exactly one of these threads returns from `sigwait` with the signal number if the signal was directed at the process.

## Related services

- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process” on page 360](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask” on page 845](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1SWT \(sigwait\) example” on page 1300](#).

# sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time

---

## Function

The `sleep` callable service suspends running of the calling thread (process) until the number of seconds specified by the parameter `Seconds` has elapsed, or until a signal is delivered to the calling thread to invoke a signal-catching function or end the thread.

## Requirements

Operation	Environment
<b>Authorization:</b>	Problem Program or Supervisor State, PSW key when the process was created (not PSW key 0)
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SLP):</b>	31-bit
<b>AMODE (BPX4SLP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Control parameters:****Environment**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SLP, (Seconds,  
              Return_value)
```

AMODE 64 callers use BPX4SLP with the same parameters.

**Parameters****Seconds**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of an unsigned fullword that contains the number of seconds for the calling thread to sleep. Because of processor delays, the calling thread can sleep slightly longer than this specified time.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of an unsigned fullword in which the sleep service returns the "remaining sleep time" value: the difference between Seconds and the number of seconds that elapsed before the thread was awakened. The return value is rounded to the nearest second. (If the thread was awakened by the ending of the elapsed time specified by Seconds, the return value is 0.) When a signal arrives and the remaining time left in the sleep is less than a half second, a value of 0 is returned.

**Usage notes**

1. The suspension can actually be longer than the requested time, because of the scheduling of other activity by the system.
2. The sleep service suspends the thread that is running for a specified number of seconds, or until a signal is delivered to the calling thread that invokes a signal-catching function or ends the thread. An unblocked signal that is received during this time prematurely "wakes up" the thread. The appropriate signal-handling function is invoked to handle the signal. When that signal-handling function returns, sleep returns immediately, even if there is "sleep time" remaining.
3. The sleep service returns a zero if it slept for the number of seconds that were specified. If the time that was specified by the Seconds parameter has not elapsed when the sleep service is interrupted because of the delivery of a signal, the sleep service returns the unslept amount of time (the requested time minus the time actually slept when the signal was delivered) in seconds. Any time that is consumed by signal-catching functions is not reflected in the value that is returned by the sleep service.
4. The following usage notes are for a SIGALRM signal that is generated by the alarm, interval timer, or kill calls during the execution of the sleep call:

- If the calling thread has SIGALRM blocked before it calls the sleep service, the sleep service does not return when SIGALRM is generated, and the SIGALRM signal is left pending when sleep returns.
  - If the calling process has SIGALRM ignored when the SIGALRM signal is generated, the sleep service does not return and the SIGALRM signal is ignored.
  - If the calling process has SIGALRM set to a signal-catching function, that function interrupts the sleep service and receives control. The sleep service returns any unslept amount of time, as it does for any other type of signal.
5. If a signal-catching function interrupts the sleep service and examines or changes the time a SIGALRM is scheduled to be generated, the action that is associated with the SIGALRM signal is the same as it is when the signal-catching function interrupts any other function.
  6. If a signal-catching function interrupts the sleep service, restores a previously saved environment, and does not return, the action that is associated with the SIGALRM signal that was saved prior to the sleep service is the same as it is when the signal-catching function interrupts any other function.
  7. When the sleep service returns, any previous alarm time that has not elapsed is restored before any signal-catcher gets control. Signal catchers can change this alarm setting. See [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#).
  8. An EC6 abend is generated when the caller's PSW key or RB state prevents signals from being delivered.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered” on page 851](#)

## Characteristics and restrictions

See [Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1413](#).

## Examples

See [“BPX1SLP \(sleep\) example” on page 1289](#).

## MVS-related information

Both the alarm service and the sleep service use the MVS STIMER macro. It is possible that two STIMER SET requests can be set by the alarm and sleep services. If the task invokes both the STIMER SET macro and the sleep service, the limit of concurrent STIMER SET requests for a task can be exceeded, which results in an abnormal end.

# smf\_record (BPX1SMF, BPX4SMF) — Write an SMF record

---

## Function

The smf\_record callable service writes an SMF record to the SMF data set. The caller must be permitted to the BPX.SMF resource profile in the FACILITY class or be APF-authorized.

The service can also be used to determine whether a particular type or subtype of SMF record is being recorded. Determining whether the type or subtype is being recorded avoids the overhead of data collection if the SMF record is not going to be recorded.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key.
<b>Dispatchable unit mode</b>	Task.
<b>Cross memory mode</b>	PASN = HASN.
<b>AMODE (BPX1SMF)</b>	31-bit.
<b>AMODE (BPX4SMF)</b>	64-bit.
<b>ASC mode</b>	Primary mode.
<b>Interrupt status</b>	Enabled for interrupts.
<b>Locks</b>	Unlocked.
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SMF, (Smf_record_type,
               Smf_record_subtype,
               Smf_record_length,
               Smf_record_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SMF with the same parameters. The Smf\_record\_address parameter is a doubleword.

## Parameters

### Smf\_record\_type

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the SMF record type. See *z/OS MVS System Management Facilities (SMF)* for information on SMF record type and SMF record layout.

### Smf\_record\_subtype

Supplied parameter.

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the SMF record subtype. See *z/OS MVS System Management Facilities (SMF)* for information about SMF record type and SMF record layout.

### Smf\_record\_length

Supplied parameter.

#### Type:

Integer

**Length:**

Fullword

The name of a field that contains the SMF record length.

**Smf\_record\_address**

Supplied parameter.

**Type:**

Address

**Length:**

Fullword (doubleword)

The name of a fullword (doubleword) that contains the starting address of the SMF record to be written, or 0. If it contains 0, SMF is tested to determine if a particular record type or subtype is being recorded.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the smf\_record service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the smf\_record service stores the return code. The smf\_record service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The smf\_record service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EINVAL</b>	The value that was specified for an operand was incorrect. The following reason code can accompany the return code: JRSMFBadRecordLength, JrSMFTypeSubtypeMismatch, or JrSMFRecordLenMismatch.
<b>ENOMEM</b>	Not enough storage is available. The following reason code can accompany the return code: JRNoStorage.
<b>EPERM</b>	The calling process is not permitted to the BPX.SMF resource in the FACILITY class and the calling processes are not APF-authorized, or the calling process is permitted to the BPX.SMF.xxx.yyy FACILITY class resource but the environment is dirty. The following reason code can accompany the return code: JRSMFNotAuthorized, JREnvDirty.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the smf\_record service stores the reason code. The smf\_record service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. To determine whether a particular type or subtype is being recorded, specify the record address as zero.
  - If the return value is zero, the type or subtype is being recorded.
  - If the return value is -1 and the return code is EMVSERR with a reason code of JRSMFNotAccepting, SMF is not recording this type or subtype.
2. When writing the SMF record, user exit IEFU84 is called if the exit is enabled. If the caller wants user exit IEFU83 to be called instead, set ThliEP\_FunctionCode to ThliEP\_SMFFlagSet and set the ThliEP\_SMFIEFU83 bit to on. These fields are mapped in the THLI control block as described in “BPXYTHLI — Thread-level information” on page 1165. z/OS UNIX callable services cannot be issued by these user exits because they are running under this syscall and nested syscalls are not allowed.
3. Because the THLIEP\_FUNCTIONCODE and the THLIEP\_MRSMFFLAGS fields may be cleared by any syscall, the values in these fields should not be expected to be retained across syscalls.
4. The smf\_record callable service supports the use of standard and extended SMF record headers.

## Related services

There are no related services.

## Characteristics and restrictions

- In order to use the smf\_record callable service, the caller must be APF-authorized or be permitted to either the BPX.SMF or the BPX.SMF.type.subtype resource profile in the FACILITY class where type is the SMF record type to be written and subtype is the SMF record subtype to be written.

To allow users to generate all SMF records, use the BPX.SMF profile. If the user is permitted to BPX.SMF, the BPX.SMF.type.subtype profiles are not checked. If the user is not permitted to generate SMF records, the exception is generated against the BPX.SMF profile.

- If the process was permitted to the BPX.SMF.type.subtype resource profile with READ access authority, the process must be program-controlled and the environment marked MUST\_STAY\_CLEAN. After the MUST\_STAY\_CLEAN state is set, any loads or execs are prevented to files that reside in uncontrolled libraries. The MUST\_STAY\_CLEAN state prevents user-written programs from writing SMF records.
- The program-controlled attribute is not required when using the BPX.SMF resource profile. If SMFUPDATE(ON) was specified in the active BPXPRMxx member, processes permitted to the BPX.SMF.type.subtype resource profile with UPDATE access authority are also exempt from the program-controlled attribute requirement.
- If you are using the BPX.SMF.type.subtype FACILITY class authorization, the type and subtype sections in the FACILITY class name do not have leading zeros. BPX.SMF.type.subtype profiles must be discrete.

Some examples:

- BPX.SMF.7.0
- BPX.SMF.119.94
- BPX.SMF.0.0

## Examples

See “[BPX1SMF \(smf\\_record\) example](#)” on page 1290.

## MVS-related information

1. See *z/OS MVS System Management Facilities (SMF)* for a list of valid SMF record types and subtypes. It also has information about SMF record layout.
2. See *Setting up the UNIX-related FACILITY and SURROGAT class profiles in z/OS UNIX System Services Planning* for a description of the BPX.SMF FACILITY class profile and how it is created.

## socket or socketpair (BPX1SOC, BPX4SOC) – Create a socket or a pair of sockets

---

### Function

The socket or socketpair callable service creates a socket or a pair of sockets for communication.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task or SRB
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SOC):</b>	31-bit task or SRB mode
<b>AMODE (BPX4SOC):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1SOC,(Domain,
               Type,
               Protocol,
               Dimension,
               Socket_vector,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SOC with the same parameters.

### Parameters

#### Domain

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the address domain requested. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the values that are defined for this field.

**Type**

Supplied and returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the type of socket that is to be created. Some of the socket types are:

**Sock#\_Stream**

Provides sequenced, two-way byte streams that are reliable and connection-oriented. They support out-of-band data.

**Sock#\_Dgram**

Provides datagrams, which are connectionless messages of a fixed maximum length whose reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

**Sock#\_Raw**

Supports AF\_INET and AF\_INET6. You must be a superuser to use this type.

Two flags are available:

**SOCK\_NONBLOCK**

Sets the O\_NONBLOCK file status flag on the open file Description. Using this flag saves extra calls to fcntl to achieve the same result.

**SOCK\_CLOEXEC**

Sets the close-on-exec (FD\_CLOEXEC) flag on the new file descriptor .

See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the values that are defined for this field.

**Protocol**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the protocol requested. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the values that are defined for this field.

**Dimension**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a field that contains the number of sockets to be returned. The value of this field determines whether the service performed is socket or socketpair. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the values that are defined for this field.

**Socket\_vector**

Supplied parameter



**Type:**

Integer

**Length:**

Doubleword

The name of a doubleword field into which a socket descriptor or pair of socket descriptors is to be stored.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the socket or socketpair service returns one of the following:

- 0, if the request is successful.
- -1 if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the socket or socketpair service stores the return code. The socket or socketpair service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The socket or socketpair service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Permission is denied.
<b>EAFNOSUPPORT</b>	The address family that was specified with the Domain parameter is not supported.
<b>EAGAIN</b>	The resource is temporarily unavailable. The following reason code can accompany the return code: JRTcpNotActive.
<b>EINVAL</b>	Dimension is not a valid value. Only 1 or 2 can be specified for this parameter. The following reason code can accompany the return code: JRInvalidParms.
<b>EIO</b>	There has been a network or transport failure. The following reason code can accompany the return code: JRPFSDead.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>EPROTONOSUPPORT</b>	The Protocol parameter is incorrect. It is not 0. The following reason code can accompany the return code: JRSocketProtocolInvalid.
<b>EPROTOTYPE</b>	The socket type is not supported by the protocol.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the socket or socketpair service stores the reason code. The socket or socketpair service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.
2. The socket callable service is invoked by specifying 1 (Sock#dim\_socket) for the Dimension parameter.
3. The socketpair callable service is invoked by specifying 2 (Sock#dim\_socketpair) for the dimension parameter. Socketpair returns 2 socket descriptors in the socket\_vector parameter. The sockets are unnamed and connected.
4. These usage notes apply for IPv6 sockets:
  - An AF\_INET6 socket may be opened only if there is at least one IPv6-capable stack active at the time of the call.
  - If a process has stack affinity under CINET, and that single stack is not IPv6-capable, the call will fail.
  - When an IPv6 socket is created through CINET, CINET creates AF\_INET6 subsockets to IPv6 stacks and IPv4 sockets to IPv4 stacks.
5. Creating a socket with stack affinity:

You can use SOCK#DIM\_SOCKETWAFFINITY and SOCK#DIM\_SOCKETPAIRWAFFINITY to create a socket or a socket pair with affinity to one specific stack under Common INET. The stack name is passed in the Socket\_vector parameter, and this field is overlaid with the output socket descriptor or descriptors when the call is successful. The name is in upper case, left-justified, and padded with blanks. This is the same name that was specified on the SUBFILESYSTYPE NAME() statement that defined this stack in BPXPRMxx.

If Common INET is not installed, the stack name is ignored. If Common INET is installed and the stack name does not match any stack configured under CINET, the call fails with a return code of EIBMBADTCPNAME.

This affinity overrides any process-level stack affinity for this one socket only.

**Note to PFS implementers:** This option is not available for the Master Socket opened as part of stack initialization.

6. UNIX domain sockets (AF\_UNIX) do not support being called in SRB-mode.

**Related services**

None.

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1SOC \(socket or socketpair\) example” on page 1291](#).

## spawn (BPX1SPN, BPX4SPN) – Spawn a process

---

### Function

The spawn callable service combines the semantics of the fork and exec callable services to create a child process to run a specified z/OS UNIX executable file.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor or problem state, any PSW key, any TCB key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SPN)</b>	31-bit
<b>AMODE (BPX4SPN)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1SPN, (Pathname_length,
               Pathname,
               Argument_count,
               Argument_length_list,
               Argument_list,
               Environment_count,
               Environment_data_length,
               Environment_data_list,
               Filedesc_count,
               Filedesc_list,
               Inherit_area_len,
               Inherit_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SPN with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

### Parameters

#### Pathname\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the path name of the file. The length of the path name can be up to 1023 bytes.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the fully qualified path name of the file that is to be run. Each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

Path names can begin with or without a slash.

- If the path name begins with a slash, it is an absolute path name; the slash refers to the root directory, and the search for the file starts at the root directory.
- If the path name does not begin with a slash, it is a relative path name; the search for the file starts at the working directory.

**Argument\_count**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the number of pointers in the lists for the Argument\_length\_list and the Argument\_list. If the program needs no arguments, specify 0.

**Argument\_length\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the 31(64)-bit address of a fullword that gives the length of an argument that is to be passed to the specified program. If the program needs no arguments, define Argument\_length\_list as the name of a fullword (doubleword) that contains 0.

If the target executable file arguments require null terminators, the arguments that are supplied to this service must include the null terminator as part of the data string and the length.

**Argument\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable that is specified by Argument\_length\_list.

The name of a list of 31(64)-bit pointers. Each pointer in the list is the 31(64)-bit address of a character string that is an argument to be passed to the specified program. Each argument is of the length that is specified by the corresponding element in the Argument\_length\_list. If the program needs no arguments, define Argument\_list as the name of a fullword (doubleword) that contains 0.

**Environment\_count**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the number of pointers in the lists for Environment\_data\_length and Environment\_data\_list. If the program needs no environment data, specify 0.

**Environment\_data\_length**

Supplied parameter

**Type:**

Structure

**Length:**

Variable

The name of a list of 31(64)-bit pointers. Each 31(64)-bit pointer in the list is the 31(64)-bit address of a fullword that gives the length of an environment variable that is to be passed to the specified program. If the program does not use environment variables, specify 0.

**Environment\_data\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable that is specified by Environment\_data\_length

The name of a list of 31(64)-bit pointers. Each 31(64)-bit pointer in the list is the 31(64)-bit address of a character string that is an environment variable to be passed to the specified program. Each environment variable is of the length that is specified by the corresponding element in Environment\_data\_length. If the program does not use environment variables, specify 0.

If the target executable file is an IBM Language Environment-enabled program, the environment variables that are supplied to this service must include the null terminator as part of the data string and length. Trailing blanks can cause the environment variable to be processed incorrectly. Each environment variable is searched for a null character; if a null character is found, the environment variable is truncated at that point.

**Filedesc\_count**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the number of file descriptors the child process is to inherit. It might take values from 0 to OPEN\_MAX. If the value is 0, all file descriptors from the parent are inherited without remapping by the child, and the filedesc\_list is ignored.

**Filedesc\_list**

Supplied parameter

**Type:**

Structure

**Length:**

Variable

The name of an array of fullword file descriptor remap values that indicate how the child's file descriptors are to be remapped from the caller's (parent's) file descriptors. Except for those

file descriptors that are designated by SPAWN\_FDCLOSED in the supplied array, each of the child's file descriptors in the range zero to Filedesc\_count-1 inherits file descriptor remap values filedesc\_list(1) to filedesc\_list(filedesc\_count) from the supplied file descriptor array. The constant SPAWN\_FDCLOSED is defined in the BPXYCONS macro.

As an example, assume that the caller supplies an array of three entries with the values 7, 5, and 4 respectively. This causes the child's file descriptor 0 to be remapped to the parent's file descriptor 7, the child's file descriptor 1 to be remapped to the parent's file descriptor 5, and the child's file descriptor 2 to be remapped to the parent's file descriptor 4.

**Inherit\_area\_len**

Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword that contains the length of the inheritance structure that is to follow. If this parameter contains a value of zero, the Inherit\_area parameter is ignored.

**Inherit\_area**

Supplied parameter

**Type:**

Structure

**Length:**

Specified by Inherit\_area\_len.

The name of a data area that contains the inheritance structure for the child process. See the BPXYINHE mapping for the details of the inheritance structure ([“BPXYINHE — Spawn Inheritance Structure” on page 1084](#)).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the spawn service returns the process ID of the newly created child process, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the spawn service stores the return code. The spawn service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The spawn service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The caller does not have appropriate permissions to run the specified file. It might lack permission to search a directory that is named in the Pathname parameter; it might lack execute permission for the file to be run; or the file to be run is not a regular file and the system cannot run files of its type.

Return_code	Explanation
<b>EAGAIN</b>	The resources that are required to let another process be created are not available now; or you have already reached the maximum number of processes or UIDs that you are allowed to create. This error is also generated if <code>_BPX_USERID</code> was specified, and the specified user name was not defined to SAF with an OMVS segment. The following reason codes can accompany the return code: JR0K, JRMaxUIDs or JRWlmWonErr.
<b>EBADF</b>	An entry in the <code>filedesc_list</code> is not a valid file descriptor; or the controlling terminal file descriptor that was specified in the inheritance structure is not valid.
<b>EINVAL</b>	<p>One or more of the following conditions were detected:</p> <ul style="list-style-type: none"> <li>• The user name that was specified on the <code>_BPX_USERID</code> environment variable has an incorrect length.</li> <li>• An attribute that was specified in the inheritance structure (BPXYINHE) is not valid or contains an unsupported value.</li> <li>• The version number that was specified for the inheritance structure (BPXYINHE) is not valid. See <a href="#">“BPXYINHE — Spawn Inheritance Structure”</a> on page 1084 for supported version numbers.</li> <li>• The inheritance structure length that was specified by the <code>Inherit_area_len</code> parameter or within the inheritance structure does not contain a length that is appropriate for the BPXYINHE version. See <a href="#">“BPXYINHE — Spawn Inheritance Structure”</a> on page 1084 for supported lengths.</li> <li>• The process group ID that was specified in the inheritance structure is less than zero or has some other unsupported value.</li> </ul> <p>The following reason codes can accompany the return code: JR0K, JRUserNameLenError, JRJsRacXtr, JRInheUserid, JRInheRegion, JRInheCPUTime, JRInheAccountData, JRInheCWD, JRInheEye, JRInheSetPgrp, JRInheVersion, JRInheLength, and JRInheMemLimit.</p>
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the <code>Filename</code> argument. This error is issued if more than 24 symbolic links are detected in the resolution of <code>Filename</code> .
<b>EMVSERR</b>	<p>If EMVSERR is accompanied by reason code <code>JrLocalSpawnNotAllowed</code>, it means that the parameters specify that a local process must be created. It also means that one or more of the current address space's attributes is to be changed. A spawn request is not allowed to change certain attributes of the current address space. A spawn request is required to create a local process when either the environment variable <code>_BPX_SHAREAS</code> is set to <code>MUST</code> (<code>_BPX_SHAREAS=MUST</code>) or when the inheritance structure in the parameter list specifies <code>InheMustBeLocal</code>. When this reason code is given, the spawn request requires a local process to be created and one or both of the following conditions is also present:</p> <ul style="list-style-type: none"> <li>• The environment variables that are specified on the call include a value for <code>_BPX_USERID</code> that does not match the current user ID.</li> <li>• The inheritance structure that is passed on the call specifies one or more of the following flags: <code>InheSetRegionSz</code>, <code>InheSetMemLimit</code>, <code>InheSetTimeLimit</code>, <code>InheSetAcctData</code>, <code>InheSetJobname</code>.</li> </ul>
<b>EMVSSAF2ERR</b>	The executable file is a set-user-ID or set-group-ID file and the file owner's UID or GID is not defined to the Security Access Facility (SAF).

Return_code	Explanation
<b>ENAMETOOLONG</b>	File_name is longer than 1023 characters, or some component of the file name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	No file name was specified, or one or more of the components of the specified Filename were not found.
<b>ENOEXEC</b>	The specified file has execute permission, but it is not in the proper format to be a process image file. Reason_code contains the loader reason code for the error.
<b>ENOMEM</b>	The new process requires more memory than is permitted by the hardware or the operating system.
<b>ENOMEM</b>	The process requires more space than is available.  The following reason code can accompany the return code: JxNSInitProcTerm, JxNamespaceNotFound.
<b>ENOTDIR</b>	A directory component of Filename is not a directory.
<b>ENOTTY</b>	The tcsetpgrp failed for the specified controlling terminal file descriptor in the inheritance structure. The failure occurred because the calling process does not have a controlling terminal, or the specified file descriptor is not associated with the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
<b>EPERM</b>	The spawn failed for one of the following reasons: <ul style="list-style-type: none"> <li>• The spawned process is not a process group leader.</li> <li>• The _BPX_USERID environment variable was specified, and the invoker does not have appropriate privileges (see <a href="#">“Authorization” on page 8</a>) to change the MVS identity.</li> <li>• The invoker does not have the appropriate privileges to change one or more of the attributes that are specified in the inheritance structure (BPXYINHE).</li> </ul> <p>The following reason codes can accompany the return code: JROK, JRNoChangeIdentity, JRInheUserid, JRInheRegion, JRInheCPUTime, and JRInheCWD.</p>
<b>ESRCH</b>	The specified process group ID in the inheritance structure is not that of a process group in the calling process's session.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the spawn service stores the reason code. The spawn service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#). For the ENOEXEC Return\_code, Reason\_code contains the loader reason code for the error:

Reason code	Explanation
<b>X'xxxx0C27'</b>	The target z/OS UNIX file is not in the correct format to be an executable file.



Reason code	Explanation
X'xxxx0C31'	The target z/OS UNIX file is built at a level that is higher than that supported by the running system.

## Usage notes for spawn

1. If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes spawn to execute a z/OS UNIX executable program file, the following conditions must be met:

- The program file and any symbolic links that comprise the path name to the file must have the proper attributes to allow the invocation.
- The z/OS UNIX executable file must be installed with an owning UID of 0 or with an owning UID equal to that of the calling set-user-ID program or have the program control extended attribute.

Any symbolic link that comprises the path name to the file or any external symbolic link must also follow these conditions, except that the program control extended attribute cannot be used because links do not have extended attributes.

Additionally, a file or link found in a file system mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Not following this setup will either cause the spawn service call to end abnormally with a EC6-xxxxE04B abend or cause the child process that is created to run the target program file to end abnormally with a EC6-xxxxE04B abend. In the latter case, spawn will return to the caller with a return code of EAGAIN and a reason code of JrForkAbend.

If a UNIX set-user-ID or set-group-ID privileged program that switched the caller's effective UID or GID invokes spawn to run a non-privileged program, the child process that is created to run the non-privileged program inherits the privilege of the set-user-ID or set-group-ID program. One exception to this is if the spawn changes the user identity for the child process. In that case, the privilege is not inherited.

The following restriction exists for sticky bit programs that have the set-user-ID or set-group-ID attributes:

- If the program is found in the MVS program search order, the MVS program name must have a BPX.STICKYSUG.program\_name resource profile defined in the RACF FACILITY class to be successfully spawned.

Failure to follow this restriction will either cause the spawn service caller to end abnormally with a EC6-xxxxE055 abend or cause the child process created to run the target program file to end abnormally with a a EC6-xxxxE055 abend. In the latter case, spawn will return to the caller with a return code of EAGAIN and a reason code of JrForkAbend.

2. Aspects of spawn processing are controlled by environment variables. The environment variables that affect spawn processing are the ones that are passed into the spawn syscall, and not the environment variables of the calling process. The environment variables of the calling process do not affect spawn processing, unless they are the same as those that are passed in the Environment\_data\_list parameter.
3. The new process (called the *child process*) inherits the following attributes from the process that calls spawn (called the *parent process*):
  - Session membership
  - Real user ID
  - Real group ID
  - Supplementary group IDs
  - Priority
  - The region size of the parent is inherited by the child, unless the INHESETREGIONSZ flag in the inheritance structure is set on to indicate that the value specified in INHEREGIONSZ is to be used

to determine the child's region size. For more information, see [What are soft limits?](#) and [Inheriting soft limits in z/OS UNIX System Services Planning](#).

- The MEMLIMIT of the parent is inherited by the child, unless the INHESETMEMLIMIT flag in the inheritance structure is set on to indicate that the value specified in INHEMEMLIMIT is to be used to determine the child's MEMLIMIT.
  - The time limit of the parent is inherited by the child, unless the INHESETTIMELIMIT flag in the inheritance structure is set on to indicate that the value specified in INHETIMELIMIT is to be used to determine the child's time limit.
  - The accounting data of the parent is inherited by the child, unless the INHESETACCTDATA flag in the inheritance structure is set on to indicate that the data pointed to by INHEACCTDATAPTR is to be used to determine the child's accounting data.
  - The current working directory (CWD) of the parent is inherited by the child, unless the INHESETCWD flag in the inheritance structure is set on to indicate that the value pointed to by INHECWDPTR is to be used to determine the child's initial current working directory.
  - The root directory of the parent is inherited by the child.
  - The file creation mask of the parent is inherited by the child, unless the INHESETUMASK flag in the inheritance structure is set on to indicate that the value specified in INHEUMASK is to be used to determine the child's file creation mask.
  - The process group ID of the parent is inherited by the child, unless the INHESETGROUP flag in the inheritance structure is set on to indicate that the value specified in INHEPGROUP is to be used to determine the child's process group. If the value in INHEPGROUP is set to INHE#NEWPGROUP, the child is placed into a new process group with a process group ID set to the child's process ID. Otherwise, the child is placed into the process group that is represented by the value that is specified in INHEPGROUP.
  - Signals that are set to be ignored in the parent are set to be ignored in the child, unless the INHESETSIGDEF flag is on and the INHESIGDEF field specifies an overriding value in the supplied inheritance structure.
  - The signal mask is inherited from the parent, unless the INHESETSIGMASK flag is set on in the inheritance structure and the INHESIGMASK field specifies an overriding value in the supplied inheritance structure.
  - The user syscall trace setting is propagated to the child process.
  - Security information from the parent's address space is propagated to the child's address space, and the child has a security environment equivalent to that of the parent, unless the \_BPX\_USERID environment variable specifies otherwise or the INHESETUSERID flag in the inheritance structure is set on, and INHEUSERID contains a user ID..
  - The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the current task are propagated to the child's address space, unless the STEPLIB environment variable specifies otherwise. This causes the child's address space to have the same MVS program search order as the calling parent task.
  - The job name of the parent is propagated to the child and appended with a numeric value in the range of 1–9, if the job name is 7 characters or less. If the job name is 8 characters, the job name is propagated as is. When a job name is appended with a numeric value, the count wraps back to 1 when it exceeds 9.
  - If the calling parent task is in a WLM enclave, the child is joined to the same WLM enclave. This allows WLM to manage the parent and child as one "business unit of work" entity for system accounting and management purposes.
  - The child is created in the same namespace as the calling process, unless a prior UNSHARE CLONE\_NEWPID or SETNS CLONE\_NEWPID was issued by the calling process.
4. The new child process has the following differences from the parent process:
- The child process has a unique process ID (PID) that does not match any active process group ID.

- The child has a different parent process ID (namely, the process ID of the process that called spawn).
- If the `filedesc_count` parameter specifies a 0 value, the child has its own copy of the parent's file descriptors, except for those files that are marked `FCTLCLOEXEC` or `FCTLCLFORK` and for directories that were opened on a call to the `opendir` service. The files marked `FCTLCLOEXEC` or `FCTLCLFORK` and open directories are not inherited by the child. If the `filedesc_count` parameter specifies a value greater than 0, the parent's file descriptors are remapped for the child as specified in the `filedesc_list` array. Those file descriptors from `filedesc_count` through `OPENMAX` in the parent are closed in the child, as are any elements in the `filedesc_list` array that are designated `SPAWN_FDCLOSED`. See the description of the `BPXYCONS` macro for the definition of the `SPAWN_FDCLOSED` constant ([“BPXYCONS — Constants used by services” on page 1062](#)). The `FCTLCLFORK` flag and `FCTLCLOEXEC` flags have no effect on inheritance when the `filedesc_list` is used to map the child's file descriptors.
- The `FCTLCLOEXEC` and `FCTLCLFORK` flags are not inherited from the parent file descriptors to the child's.
- If the `INHESETTCPGRP` flag is set in the inheritance structure, `INHECTLTTFD` must be set to the file descriptor that is associated with the controlling terminal for this session. The foreground process group for this session is set to the PGID of this child process, thus placing the child process in the foreground process group. (This is done by issuing a `tcsetpgrp()` syscall as a part of spawn processing.)
- If `INHESETTCPGRP` is not set, the foreground process group of the session remains unchanged.
- If the `INHESETCWD` flag is set on in the inheritance structure, the child's initial working directory is set to the directory path described by `INHECWDPTR` (pointer) and `INHECWDLEN` (length), if the caller has appropriate privileges.
- If the `INHESETUMASK` flag is set on in the inheritance structure, the child's file mode creation mask (`umask`) is set to the value in `INHEUMASK`, if the caller has appropriate privileges.
- If the `INHESETREGIONSZ` flag is set on in the inheritance structure, the child's region size is set to the value in `INHEREGIONSZ`, if the caller has appropriate privileges.
- If the `INHESETTIMELIMIT` flag is set on in the inheritance structure, the child's CPU time limit is set to the value in `INHETIMELIMIT`, if the caller has appropriate privileges.
- If the `INHESETACCTDATA` flag is set on in the inheritance structure, the child's accounting data is set to the value specified by `INHEACCTDATAPTR` (pointer) and `INHEACCTDATALEN` (length).
- If the `INHESETMEMLIMIT` flag is set on in the inheritance structure, the child's `MEMLIMIT` is set to the value indicated in `INHEMEMLIMIT`, if either the caller has appropriate privileges or `INHEMEMLIMIT` is less than the current hard limit of the target address space.
- The process and system utilization times for the child are set to zero.
- Any file locks that were previously set by the parent are not inherited by the child.
- The child process has no alarms or interval timers set. (This is similar to the results of a call to the alarm service with `Wait_time` specified as zero.)
- The child has no pending signals.
- The child gets a new process image to run the executable file that is not a copy of the parent's.
- Signals that are set to be caught are reset to their default action.
- Memory mappings that are established by the parent with the `shmat` or `mmap` services are not inherited by the child.
- The semaphore adjustment values, `semadj`, are set to zero.
- If the `SSTFNOSUID` bit is set for the file system that contains the new process image file, the effective user ID, effective group ID, saved set-user-ID and saved set-group\_ID are unchanged in the new process image. Otherwise, if the `setuid` bit of the new process image file is set, the effective user ID of the new process image is set to the owner id of the new process image file. Similarly, if the `setgid` bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID, real group ID,

and supplementary group IDs of the new process image remains the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID) for use by the setuid and setgid functions.

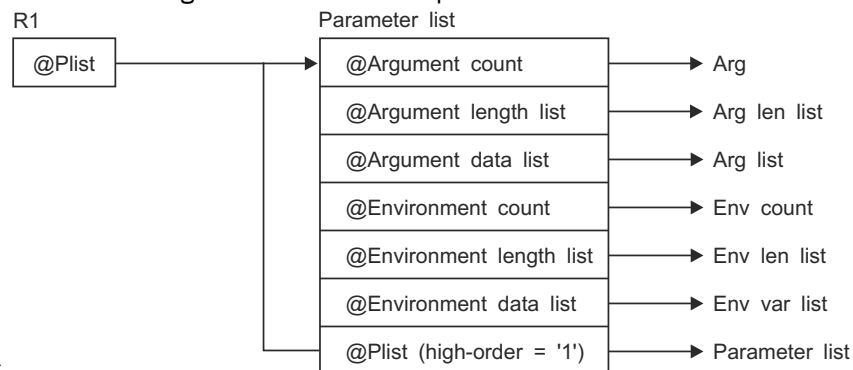
5. When run in its own address space, the executable file that is to be run receives control with the following attributes:

- Problem program state
- TCB and PSW key 8
- AMODE=31(64), taken from the executable
- Primary ASC mode

When run in the same address space as the caller of spawn:

- Problem program state
- TCB and PSW key equal to the TCB key of the caller of BPX1SPN/BPX4SPN
- AMODE=31(64), taken from the executable
- Primary ASC mode

6. The information that the service passes to the executable file to be run is a parameter list, which is pointed to by register 1. In 31-bit mode, the parameter lists contains a 4-byte addresses; in 64-bit mode the parameter list contains a 64-byte addresses. In 31-bit mode, in the last parameter address the high-order bit is 1. The parameter list consists of the following parameter



addresses.

For AMODE 31 callers, the high-order bit in the last parameter address is 1. For AMODE 64 callers, the high-order bit is part of the 64-bit address. There are always *n* parameters, passed with no end-of-parameter-list indicator.

The last parameter that spawn passes to the executable file identifies the caller of the file as the exec or spawn service.

7. To support the creation and propagation of a STEPLIB environment to the new process image, the spawn service allows for the specification of a STEPLIB environment variable. The following table lists the accepted values for the STEPLIB environment variable, and the actions that are taken for each value:

Value	Action taken
STEPLIB=NONE	No STEPLIB DD is created for the new process image.
STEPLIB=CURRENT	The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
STEPLIB=Dsn1:Dsn2:,...DsnN	The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.

Value	Action taken
	<p><b>Note:</b> The actual name of the DD is not STEPLIB. Instead, it is a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order specified. The specified data sets must follow standard MVS data set naming conventions. If they do not follow the standard naming conventions, they are ignored. The data sets are also ignored if they do follow the standard naming conventions but these situations exist:</p> <ul style="list-style-type: none"> <li>• The caller does not have the proper security access to a data set.</li> <li>• A data set is uncataloged or is not in load library format.</li> </ul> <p>The system will also ignore the data set and continue allocating the other data sets in these situations:</p> <ul style="list-style-type: none"> <li>• If the system cannot acquire a shared ENQueue on the data set name because another job has ENQueued it exclusively.</li> <li>• If the specified data set does not exist. This case is similar but not strictly equal to the case of uncataloged data sets.</li> </ul> <p>Because the data sets are ignored, the executable file might run without the proper STEPLIB environment. If a data set is ignored due to improper security access, an X'913' abend is generated. The dump for this abend can be suppressed by your installation</p> <p>If the program to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For more information about the sanction list, see <a href="#">Using sanction lists in z/OS UNIX System Services Planning</a>. For information about STEPLIB performance considerations, see <a href="#">Controlling use of STEPLIBs in z/OS UNIX System Services Planning</a>.</p>

If the STEPLIB environment variable is not specified, the default behavior of the spawn service is the same as if STEPLIB=CURRENT were specified.

If the program that is to be invoked is a set-user-ID or set-group-ID file, and the user-ID or group-ID of the file is different from that of the current process image, the data sets that are to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. See [Using sanction lists in z/OS UNIX System Services Planning](#) for detailed information about the sanction list. See [Tuning performance in z/OS UNIX System Services Planning](#) for information about STEPLIB performance considerations.

8. For performance reasons, the spawn service is allowed to create a new image under a specific user ID that is different from that of the invoker. When an invoker with appropriate privileges specifies a user name on the `_BPX_USERID` environment variable or in the inheritance structure (INHEUSERID), the resulting image runs under the associated MVS user identity. This service allows `spawn()` to replace the sequence of functions `fork()`, `setgid()`, `initgroups()`, `setuid()`, and `exec()`. The following rules apply to `spawn()`s with user name changes (using either the `_BPX_USERID` environment variable or the inheritance structure):
  - a. `_BPX_USERID` must be a valid 1-8 character XPG4-compliant name with a defined OMVS segment. An incorrect username length results in a failure of the `spawn()` request with an `EINVAL` and `JRUserNameLenError`. An undefined username results in an `ERRNOJR` of `JRJsRacXtr`; an

incompletely defined OE username results in an ERRNO of EMVSSAF2ERR and ERRNOJR of the propagated SAF return code and reason code.

- b. If the creation of the new address space with the new user identity puts the system over the limit of MAXUIDs, spawn() fails with an ERRNO of EAGAIN and an ERRNOJR of JRMaxUIDs.
  - c. Authorization to change the username is the same as for the setuid() function. If the caller is not authorized, spawn() fails with an ERRNO of EPERM and an ERRNOJR of JRNoChangeIdentity (\_BPX\_USERID) or an ERRNOJR of JRInheUserid (inheritance structure).
  - d. If \_BPX\_SHAREAS is active and an identity change is called for, \_BPX\_SHAREAS is ignored and a new address space is created for the new image.
9. To allow the caller to control whether the spawned child process runs in a separate address space from the parent address space or in the same address space, the spawn service allows for the specification of the \_BPX\_SHAREAS environment variable. The following are the accepted values for the \_BPX\_SHAREAS environment variable, and the actions that are taken for each value:

Value	Action taken
_BPX_SHAREAS=YES	<p>Indicates that the child process that is to be created is to run in the same address space as the parent. In the following circumstances, the _BPX_SHAREAS=YES value cannot be honored, and the child process is created in its own address space:</p> <ul style="list-style-type: none"> <li>• If the program to be run is a set-user-ID or set-group-ID program that would cause the effective user-ID or group-ID of the child process to be different from that of the parent process.</li> <li>• If the program to be run is an APF-authorized z/OS UNIX or MVS program and the caller is not running APF-authorized.</li> <li>• If the program to be run is an unauthorized z/OS UNIX or MVS program and the caller is running APF-authorized.</li> <li>• If the specified file name represents an external link or a sticky bit file. If, however, the program that is to be run is a shell script and _BPX_SPAWN_SCRIPT=YES is set, the process runs in the same address space.</li> <li>• If the parent's address space lacks the necessary resources to create another process within the address space.</li> </ul> <p>Only one local spawned process per TSO address space is supported at a given time. This restriction reduces conflict among multiple shells that are running in the same address space.</p>
_BPX_SHAREAS=MUST	<p>Indicates that the child process that is to be created must run in the same address space as the parent, or the spawn request will fail. In the following circumstances, the _BPX_SHAREAS=MUST value cannot be honored, and the spawn invocation fails:</p> <ul style="list-style-type: none"> <li>• If the program to be run is a set-user-ID or set-group-ID program that would cause the effective user ID or group ID of the child process to be different from that of the parent process.</li> <li>• If the program to be run is an APF-authorized z/OS UNIX or MVS program and the caller is not running APF-authorized.</li> <li>• If the program to be run is an unauthorized z/OS UNIX or MVS program and the caller is running APF-authorized.</li> <li>• If the parent's address space lacks the necessary resources to create another process within the address space.</li> <li>• If the list of environment variables specifies a value for _BPX_USERID that is different from the current user ID.</li> </ul>

Value	Action taken
	<ul style="list-style-type: none"> <li>If the inheritance structure sets any of the following flag bits: InheSetRegionSz, InheSetMemLimit, InheSetTimeLimit, InheSetAcctData, InheSetJobname.</li> </ul>
_BPX_SHAREAS=REUSE	Specifying _BPX_SHAREAS=REUSE is equivalent to specifying _BPX_SHAREAS=YES.
_BPX_SHAREAS=NO	Indicates that the child process that is to be created is to run in a separate address space from the parent's address space. This is the default behavior for the spawn service if the _BPX_SHAREAS environment variable is not specified, or if it contains an unsupported value.

10. When the parent and child processes are sharing address space, special consideration must be given to the resources that are shared in this environment. These considerations include, but are not limited to, the following ones:
- The parent and child share data set allocations, and must coordinate usage of these allocations. Programs that have special ddname allocation requirements should not be run in this shared environment.
  - The parent and child share private area storage; they should be careful not to overreach their own storage bounds, and together they must not exceed the region size of the address space.
  - A prior loaded copy of a z/OS UNIX program is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS XCTL service, with the following exceptions:
    - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
    - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the z/OS UNIX program found is in storage that is modifiable by the caller, the prior copy is not reused.
  - If the specified file name represents an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is used only if the name is 8 characters or less; otherwise the caller receives an error from the spawn service. For a sticky bit program, the file name is used if it is 8 characters or less. Otherwise, the program is loaded from the z/OS UNIX file system.
  - The program that is being spawned should have the APF extended attribute turned on and should be linked AC=1. DLLs that are loaded by APF-authorized applications should have the APF extended attribute set on and should be linked AC=0.
11. To allow the caller to control whether the spawn service is to treat the specified file as a shell script if it is found not to be in the correct format to be a process image file, the spawn service allows the specification of the \_BPX\_SPAWN\_SCRIPT environment variable. The following are the accepted values for the \_BPX\_SPAWN\_SCRIPT environment variable, and the actions that are taken for each value:

Value	Action taken
_BPX_SPAWN_SCRIPT=YES	<p>Treats the specified file as a shell script if it is not in the correct format to be a process image file, does not contain the "magic number" (!) in the first line, and is not a REXX exec. In this case, the spawn service behaves as follows:</p> <ul style="list-style-type: none"> <li>The z/OS shell executable file that is current in the caller's environment is executed to run the specified file as a shell script. The path name for the shell executable file is determined by extracting the path name from the SHELL environment variable,</li> </ul>

Value	Action taken
	<p>if the SHELL variable is present in the environment data list that is supplied to spawn. If it is not present, the default path name of /bin/sh is used as the shell executable path name.</p> <ul style="list-style-type: none"> <li>The argument data list and argument length list that are passed to the shell executable file are to contain the following argument data and corresponding argument data lengths: <ul style="list-style-type: none"> <li>The shell path name, terminated by a null character (X'00').</li> <li>The string -S, terminated by a null character (X'00').</li> <li>The string –, terminated by a null character (X'00').</li> </ul> </li> </ul> <p>The fourth through the last arguments in the list are to contain the list of arguments that are specified by the caller of the spawn service.</p>
_BPX_SPAWN_SCRIPT=NO	<p>The specified file is not treated as a shell script if it is found not to be in the correct format to be a process image file, does not to contain the file magic number (#!), and is not a REXX exec. In this case, the spawn service fails and returns the return code ENOEXEC to the caller of spawn. This action is the default behavior for the spawn service if the _BPX_SPAWN_SCRIPT environment variable is not specified, or if it contains an unsupported value.</p>

12. If the specified z/OS UNIX file is not in the correct format to be an executable, but contains the magic number (!) in the first line, the program that is specified in the magic number header is executed. The expected format of the magic number header is as follows:

```
#! Path String
```

#! is the file magic number. It identifies the first line of the file as a special header that contains the name of the program to be run and any argument data to be supplied to it.

The Path parameter specifies the path name of the file that is to be run. It is separated by blank or tab characters from the #! characters, or can immediately follow the characters.

The String parameter is an optional character string that can be used to pass options to a target command interpreter (shell) that is to run the script. It must be separated from the Path parameter by tab or blank characters, and cannot itself contain tab or blank characters.

The argument data list and argument length list that are passed to the magic number file are to contain the following argument data and corresponding argument data lengths:

- The magic number path name, ended by a null character (X'00').
- The string, if one is supplied, ended by a null character (X'00').

The remaining arguments in the list are to contain the list of arguments that are specified by the caller of the spawn service.

If the path name that is specified in the magic number header cannot be executed for some reason, the spawn request fails with return code ENOEXEC, regardless of the error. ENOEXEC is returned for compatibility purposes, so that existing scripts can continue to run successfully when invoked from an application such as a command interpreter (shell). The reason code indicates the exact reason the magic number file could not be executed.

13. If the target executable program is enabled by Language Environment, the environment variables that are supplied to the service must include the null terminator as part of the string and length.
14. If the \_BPX\_PTRACE\_ATTACH environment variable is set to YES, the target executable program is loaded into user-modifiable storage to allow subsequent debugging. Any additional programs that are



loaded into storage during the execution of the target program are also loaded into user-modifiable storage, except for modules that are loaded from the LPA.

15. The `_BPXK_MDUMP` environment variable can be used to specify where a `SYSDUMP` is to be written. The following table lists the allowable values:

Value	Description
OFF	The dump is written to the current directory. This is the default. This dump is only written if the user allocates a <code>SYSDUMP</code> data set for the TSO/E session. The system creates a file that is named <code>coredump.pid</code> in the user's working directory (where <i>pid</i> is the process ID for the process that is being dumped) and writes the core dump ( <code>SYSDUMP</code> ) in hexadecimal format.
MVS data set name	The dump is written to an MVS data set. The data set name must be fully qualified and can be up to 44 characters. It can be specified in uppercase, lowercase, or both; it is folded to uppercase.
z/OS UNIX file name	The dump is written to a z/OS UNIX file. The file name can be up to 1024 characters and must begin with a slash. The slash refers to the root directory, in which the file is created.

16. The `_BPXK_JOBLOG` environment variable can be used to specify that WTO messages are to be written to an open z/OS UNIX job log file. The following table lists the allowable values:

Value	Description
<i>nn</i>	Job log messages are written to open file descriptor <i>nn</i> .
STDERR	Job log messages are written to the standard error file descriptor, 2.
NONE	Job log messages are not written. NONE is the default.

The file that is used to capture messages can be changed at any time by calling the `oe_env_np` service and specifying `_BPXK_JOBLOG` with a different file descriptor.

Message capturing is turned off if the specified file descriptor is marked for close on a fork or exec.

Message capturing is process-related. All threads under a given process share the same job log file. Message capturing may be initiated by any thread under that process.

Multiple processes in a single address space can each have different files active as the `JOBLOG` file; some or all of them can share the same file; and some processes can have message capturing active while others do not.

When the file that is used as a job log is shared by several processes (for example, by a parent and child), the file should be opened for append. Failure to do this causes unpredictable results.

Only files that can be represented by file descriptors may be used as job log files; MVS data sets are not supported.

Message capturing is propagated on a `fork()` or `spawn()`. If a file descriptor is specified, the physical file must be the same in order for message capturing to continue in the forked or spawned process. If `STDERR` is specified, the file descriptor may be remapped to a different physical file.

Message capturing may be overridden on `exec()` or `spawn()` by specifying the `_BPXK_JOBLOG` environment variable as a parameter on the `exec()` or `spawn()`.

Message capturing only works in forked (BPXAS) address spaces.

This is not true job log support: messages that would normally go to the `JESYSMSG` data set are captured, but messages that go to `JESMSGGLG` are not.

17. When the INHESETUSERID or INHESETACCTDATA flags are set on in the inheritance structure, the corresponding environment variables for user name (\_BPX\_USERID) or accounting data (\_BPX\_ACCT\_DATA) are ignored.
18. Depending on the attributes and values that are specified in the inheritance structure (BPXYINHE), the caller is required to have different levels of authorization. The following table defines the specific authorization that is required for each attribute. Inheritance structure attributes that are not listed in the table do not require authorization.

<b>BPXYINHE field</b>	<b>Authority required</b>
<b>INHEUSERID</b>	The caller must have daemon authority. See <a href="#">Giving daemon authority to vendor-written programs in z/OS UNIX System Services Planning</a> for information about setting up daemon authority for a user.
<b>INHEREGIONSZ</b>	When the new region size is smaller than the caller's current hard limit for RLIMIT_AS, no authorization is required. To exceed the current hard limit, the caller must have superuser authority (UID=0), or the spawn function will fail.
<b>INHETIMELIMIT</b>	When the new CPU time limit is less than the caller's current hard limit for RLIMIT_CPU, no authorization is required. To exceed the current hard limit, the caller must have superuser authority (UID=0), or the spawn function will fail.
<b>INHEUMASK</b>	The caller must have superuser authority to set the child's file mode creation mask, or the spawn function will fail.
<b>INHECWD</b>	INHECWD - The child's current working directory can be changed if the caller has the correct permissions to it. If not, then the spawn function will fail with EACCES - Permission is denied.

19. If the INHESETDEBUGENV flag in the inheritance structure is set on, the target program is under the control of the debugger process.
20. If the INHEMUSTBELOCAL flag in the inheritance structure is set on, the program must run in the same address space as the caller, or the spawn invocation will fail. (This flag causes the same behavior as \_BPX\_SHAREAS=MUST.) See Usage Note 8b for conditions that will cause the spawn invocation to fail when the INHEMUSTBELOCAL flag is set on.
21. If the BPXK\_SIGDANGER environment variable is set to YES, the process will receive a SIGDANGER signal rather than a SIGTERM signal when an OMVS shutdown is initiated. This may be advantageous for an application that uses the SIGTERM signal for other purposes.
22. The \_BPX\_UNLIMITED\_OUTPUT environment variable can be used to specify that default installation limits for SYSOUT output are to be overridden. This environment variable is only processed when it is specified for a non-local spawn. These are the supported values:

<b>Value</b>	<b>Description</b>
YES	The SYSOUT job limits (BYTES, CARDS, LINES and PAGES) is set to the maximum values and an action of WARNING. This effectively allows unlimited output for the job that is associated with the newly spawned child.
NO	The installation defaults will be used for job output limits. This behavior is the default. Any other value results in the installation defaults being used.
z/OS UNIX file name	The dump is to be written to a z/OS UNIX file. The file name can be up to 1024 characters and must begin with a slash. The slash refers to the root directory, in which the file is created.

For non-local spawn to process the \_BPX\_UNLIMITED\_OUTPUT environment variable, the caller must have appropriate privileges:

- a. Be a superuser (UID=0) or
  - b. Be permitted to the BPX.UNLIMITED.OUTPUT resource profile in the FACILITY class with at least READ access.
23. When the spawn service is called from a \_\_login process in a multiprocess, multiuser environment (environment variable \_BPX\_SHAREAS=YES is in effect), the newly created child process will have the MVS identity of the address space and the POSIX permissions of its parent process. It is expected that the application program will immediately call the \_\_login service to establish a proper process-level identity in the child process.
24. The \_BPXK\_INITTAB\_RESPAWN environment variable specifies whether a process is to be dynamically started with the respawn attribute. The following are the allowable values:

Value	Description
YES	Specifies that a process is to be started with the respawn attribute. Setting this to YES after the process has started does not affect the setting of the respawn attribute.
NO	Disables the respawn capability of the process. The process is not respawned when it ends.

25. When the executable file to be run is a REXX exec, the first argument should be the path name of the REXX exec. Subsequent arguments for the exec can follow this argument.
26. The \_BPXK\_INITTAB\_RESPAWN attribute is only allowed for non-local spawns. The \_BPXK\_INITTAB\_RESPAWN setting is ignored if the process is spawned in the same address space. Setting \_BPX\_SHAREAS=YES, \_BPX\_SHAREAS=MUST, or INHEMUSTBELOCAL causes the \_BPXK\_INITTAB\_RESPAWN option to be ignored on the spawn call.
27. z/OS UNIX sets a default message class of “A” for all forked or spawned processes. Unlike JES, z/OS UNIX does not have a method for accepting a user-supplied default message class, and a default had to be supplied to the converter interpreter. Message class A was chosen as the default for BPXAS initiators. This default value cannot be changed dynamically. The MSGCLASS for the joblog (JESMSG LG, JESJCL, JESYSMSG) is set to class A before the fork or spawn that associates the process with the BPXAS initiator is begun.
28. The \_BPX\_JOBNAME environment variable can be used to change the job name of the new process image. The job name change is allowed only if the invoker has appropriate privileges (see “Authorization” on page 8). If these conditions are not met, the environment variable is ignored. Accepted values are strings of 1–8 alphanumeric characters. Incorrect specifications are ignored.
- A job name can be specified in the INHE structure. If both the INHE JOBNAME and environment variable \_BPX\_JOBNAME are specified, the value in the INHE structure is used.
  - If \_BPX\_JOBNAME is specified and \_BPX\_SHAREAS is set to YES, then \_BPX\_JOBNAME is ignored.
29. If the caller specifies \_BPXK\_DISABLE\_SHLIB=YES then future loadhfs() and loadhfs\_extended() system calls will ignore the st\_sharelib attribute and load the program into private storage. If the caller specifies NO (the default) then normal system shared library processing takes place. For more information about \_BPXK\_DISABLE\_SHLIB, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).
30. If the specified file name resolves to an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is used only if the name is 8 characters or less; otherwise the caller receives an error from the spawn service. For a sticky bit program, the file name is used if it is 8 characters or less. Otherwise, the program is loaded from the z/OS UNIX file system. If the specified sticky bit file or link resolves to an MVS program link-edited AC=1 in an APF-authorized library, the attributes of the sticky bit file or external link must be set up properly to allow this type of invocation. For a sticky bit file, it must be installed with an owning UID of 0 or with the APF extended attribute. The owning UID of 0 requirement would also apply to a symbolic link that resolves to the sticky bit file. For an external link, it must be installed with an owning UID of 0. A sticky bit a file with the APF extended attribute is not allowed if found in a file system that is mounted as NOSETUID. If the specified file name represents a symbolic link to

a sticky bit file that has the set-user-id attribute, the symbolic link must have an owning uid of 0 or an owning uid equal to that of the sticky bit file. If the sticky bit file has the set-group-id attribute, the symbolic link must have an owning uid of 0 or an owning GID equal to that of the sticky bit file. A file or link that is found in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Failure to follow this set up will cause the child process created to run the MVS program to end abnormally with a EC6-xxxxC04A abend when the MVS program is invoked via the spawn service.

## Related services

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory ” on page 101](#)
- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors” on page 216](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“sigpending \(BPX1SIP, BPX4SIP\) — Examine pending signals ” on page 843](#)
- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ” on page 777](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ” on page 893](#)
- [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs” on page 800](#)
- [“tcsetpgrp \(BPX1TSP, BPX4TSP\) — Set the foreground process group ID ” on page 944](#)
- [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask” on page 960](#)

## Characteristics and restrictions

1. Executing a program from z/OS UNIX causes the program environment to become uncontrolled, unless the program is identified as program controlled. (That is, unless the ST\_PROGCTL attribute is ON for the z/OS UNIX program file). Running a z/OS UNIX program with the ST\_PROGCTL attribute set to OFF prevents future invocations of authorized programs like program access to data sets (PADS) programs. These are programs that are given special authorization by the installation and by the installed security product (such as RACF) to read or write to protected data sets. In addition, PADS programs should not attempt to load programs from z/OS UNIX with the ST\_PROGCTL attribute OFF, because these programs are considered uncontrolled and could have been modified by users that do not have the same level of authorization as the PADS program.
2. The \_BPXK\_TIMEOUT environment variable is processed for this callable service. It allows applications to specify whether a process that is waiting on terminal input in the shell should be timed out. For more information about the \_BPXK\_TIMEOUT environment variable, see [Commonly used environment variables in z/OS UNIX System Services Planning](#).

## Examples

See [“BPX1SPN \(spawn\) example” on page 1293](#).

# srx\_np (BPX1SRX, BPX4SRX) — Send or receive CSM buffers on a socket

---

## Function

The srx\_np callable service sends or receives data on a socket using CSM (Communications Storage Manager) buffers.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task SRB - AF_INET/AF_INET6 socket support only
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SRX):</b>	31-bit task or SRB mode
<b>AMODE (BPX4SRX):</b>	64-bit SRB mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SRX, (Socket_descriptor,
               Direction,
               Msghdrx_length,
               Msghdrx,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SRX with the same parameters. All addresses in the Msghdrx structure are 31-bit pointers.

## Parameters

### Socket\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the socket file descriptor for which the srx\_np service is requested.

### Direction

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains a value that indicates the operation requested:

- MSGX\_SEND or 0, for a send operation
- MSGX\_RECV or 1, for a receive operation

### Msghdrx\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the length of the Msghdrx parameter.

Before the data structures are built for the first time, you can use a value of 0 in this field to determine whether the operation is supported on a given socket. If the operation is supported, a Return\_value of 0 is returned. If the operation is not supported, a Return\_value of -1 with a Return\_code of ENOSYS is returned.

### Msghdrx

Supplied and returned parameter

#### Type:

Structure

#### Length:

Length of MSGX from BPXYMSGX

The name of the MSGX structure that contains the information for this operation. See the usage notes and the BPXYMSGX macro ("[BPXYMSGX — Map the message header](#)" on page 1109) for more information about the MSGX structure.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the srx\_np service returns one of the following:

- The number of bytes that were sent or received from the buffers, if the request is successful.
- -1, if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the srx\_np service stores the return code. The srx\_np service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The srx\_np service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>ENOSYS</b>	This function is not supported on the specified socket.
<b>EAFNOSUPPORT</b>	The address family that was specified in the message header is not the same as the address family that owns the socket.
<b>EBADF</b>	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	The connection was reset by a peer. The following reason code can accompany the return code: JRSockNotCon.

Return_code	Explanation
<b>EINTR</b>	A signal interrupted the srx_np service before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	An input parameter was incorrect. The following reason codes can accompany the return code: JRInvalidMsgh, JRSocketCallParmError, and JRSockNoName.
<b>EMSGSIZE</b>	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
<b>EFAULT</b>	An address that was passed pointed to storage that could not be accessed.
<b>ENOTCONN</b>	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
<b>EPIPE</b>	An attempt was made to send a message to a socket that is shut down or closed. This error also generates a SIGPIPE signal. The following reason code can accompany the return code: JRSocketClosed.
<b>EWouldBlock</b>	For a receive operation, the socket is marked nonblocking and there is no data available to be read, or the SO_RCVTIMEO timeout value was reached before data was available. The following reason codes can accompany the return code: JRWouldBlock, JrTimeOut.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the srx\_np service stores the reason code. The srx\_np service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.
2. The Communications Storage Manager (CSM) provides a facility that allows programs to avoid data moves on a communications session by transferring buffer ownership instead of copying the buffer contents.

The srx\_np service provides a way to send these buffers on a socket session. It is assumed that the application has its own interactions with CSM that allow it to obtain and free these buffers independently from the srx\_np service. CSM is restricted to authorized programs, and the buffers are in key 6 storage. The srx\_np service, however, may be invoked from problem state or authorized programs. All parameters are in local application storage and the caller's key.

For more information about CSM, see *z/OS Communications Server: CSM Guide*.

3. The general flow for using this service is as follows:

When sending:

- a. The application obtains CSM buffers and fills them with the data to be sent. The collection of one or more CSM data buffers is described with an IOVX array that may be built in application storage or in

another CSM buffer. This whole structure and operational characteristics are specified in a Msghdrx structure, which is passed to the srx\_np callable service.

- b. The request is passed on to the transport, such as TCP/IP, for the specified socket. If the socket transport does not support CSM buffers, the call fails with ENOSYS. A specific socket can be tested for support before the buffers are built, by specifying 0 for Msghdrx\_length. If CSM buffers are not supported, the data will have to be sent with standard services, such as send (BPX1SND, BPX4SND) or writev (BPX1WRV, BPX4WRV). CSM buffers could be used on the standard services, but they would be treated as application buffers, and the application would retain ownership and be responsible for freeing the buffers.
- c. The socket transport transfers ownership of the CSM data buffers, not the IOVX, to itself and passes them along to the communications adapter. Ownership of the IOVX buffer, if it is a CSM buffer, remains with the application.
- d. The communications adapter transfers buffer ownership to itself and transmits the data.
- e. When the I/O is complete, the adapter issues CSM deallocates for the buffers.
- f. CSM puts the buffers back into its global free pool.
- g. When control returns to the application after the srx\_np call, it no longer owns the buffers and must not reference them again.

When receiving:

- a. Inbound data is received into CSM buffers obtained by the communications adapter.
- b. These buffers are passed up to the socket transport, who assumes ownership.
- c. The application calls srx\_np to receive. A Msghdrx structure is passed that may contain some control information, but that does not specify any buffers or an IOVX array.
- d. The socket transport builds an IOVX array to describe the inbound data buffers that have been accumulated. This array is itself in a CSM buffer.

If data has not arrived yet, the request is suspended or failed with EWOULDBLOCK, as for any other socket receive type of operation.

When data is to be returned to the application, the transport assigns ownership of the CSM buffers to the application, and the application's Msghdrx structure is filled in with a description of the IOVX array buffer.

- e. When control returns to the application after the srx\_np call, it has ownership of the CSM buffers and may process the data that has been received.
- f. When the application has finished with the buffers and the IOVX array, it issues CSM deallocates for them.
- g. CSM puts the buffers back into its global free pool.

4. For a receive operation, Msghdrx contains the following fields:

Field	Description
<b>MsgxNamePtr</b>	A pointer to a sockaddr buffer in which the system returns the source address of the data that is received.  This field is optional. If it is not used, MsgxNamePtr or MsgxnameLen should be zero.
<b>MsgxNameLen</b>	The length of the sockaddr buffer that is pointed to by MsgxNamePtr.



Field	Description
<b>MsgxIovx</b>	<p>An IVTBUFL structure in which the system describes the CSM buffer containing the IOVX array being returned for this request. This CSM buffer is obtained by the system and freed by the calling application.</p> <p>The IOVX array contains IVTBUFL structures, each of which describes a CSM data buffer that contains the data received by this request. The CSM data buffers that are used by this service are obtained by the system and freed by the caller.</p>
<b>MsgxMsgFlags</b>	MSG_* flags for this operation. Refer to <a href="#">“BPXYMSGF — Map the message flags”</a> on page 1107.
<b>MsgxFlags</b>	<p>Control flags:</p> <ul style="list-style-type: none"> <li>• MSGX_CECSA, indicating that the CSM buffers should be obtained from ECSA</li> <li>• MSGX_CDSPACE, indicating that the CSM buffers should be obtained from one of the CSM data spaces</li> </ul> <p>If neither flag is specified, the application can handle CSM buffers in either ECSA or a data space.</p>
<b>MsgxDataLen</b>	<p>The maximum or minimum amount of data that is to be received:</p> <ul style="list-style-type: none"> <li>• When MSG_WAITALL is off, MsgxDataLen specifies the maximum amount of data that the caller wants to receive.</li> <li>• When MSG_WAITALL is on, MsgxDataLen specifies the minimum amount of data that the caller wants to receive.</li> </ul> <p>You can use this value to control the amount of data that is received, in the same way that you use the Buffer_length parameter of the recv service.</p> <p>If this field is 0, the receive operation completes as soon as the first block of data is available, and whatever data is available is returned.</p> <p>If the receive operation cannot be completed immediately, the application blocks or receives an EWOULDBLOCK error, depending on its blocking state.</p>
<b>MsgxTcb</b>	<p>The TCB address of a task with which the CSM storage is to be associated. By default the storage is associated with the calling task.</p> <p>This field is optional, and should be 0 if not specified.</p>

5. For a send operation, Msghdrx contains the following fields:

Field	Description
<b>MsgxNamePtr</b>	<p>A pointer to a sockaddr buffer that contains the destination address for the send operation.</p> <p>This field is optional. If it is not used, MsgxNamePtr or MsgxNameLen should be 0.</p>
<b>MsgxNameLen</b>	The length of the sockaddr buffer that is pointed to by MsgxNamePtr.

Field	Description
<b>MsgxIovx</b>	An IVTBUFL structure that describes the buffer containing the IOVX array. This buffer may be a CSM buffer, or it may be in the caller's storage. Ownership of a CSM buffer used for the IOVX array remains with the application.  The IOVX array contains IVTBUFL structures, each of which describes a CSM data buffer that contains the data to be sent. The CSM data buffers that are used by this service are obtained by the caller and freed by the system.
<b>MsgxMsgFlags</b>	MSG_* flags for this operation. Refer to <a href="#">“BPXYMSGF — Map the message flags”</a> on page 1107.
<b>MsgxIVTBUFOffset</b>	The returned offset of the IOVX array entry for the first CSM data buffer that the application still owns. After a successful send, this value is equal to the length of the IOVX array. If this value is zero, no buffers were taken.
<b>MsgxErrIovx</b>	The offset of the IOVX array entry that is in error.  This field and MsgxErrData are returned only when there is an error that is specifically related to one of the IOVX entries or their associated buffers. Refer to the Return_code and Reason_code for details on the error.
<b>MsgxErrData</b>	The amount of data that has been sent successfully from the buffer that is indicated by MsgxErrIovx.

MsgxErrIovx and MsgxErrData should only be examined when the request completes with a Return\_value of -1, or when the amount of data sent is less than the amount of data that was requested to be sent.

6. A C header, BPXYSRXH, is available which contains a C structure for the Msghdrx and a prototype for srx\_np. With this header and the IVTBUFL C header, you can send and receive CSM buffers from C programs.

Note, however, that this program would simply be making C calls to the srx\_np callable service, and not making normal C functional references. In particular, the return value and errno value would be returned in explicit calling parameters, rather than in the standard C method.

7. The socket may be connected or unconnected.
8. Consult the documentation for the TCP/IP stack that is being used for support for this function.
9. Some common INET considerations:

When the socket is associated to a specific transport, the requests are accepted or rejected based on that transport's support for CSM buffers. A socket becomes associated with a specific transport by being a connected stream socket, bound to a specific IP address, or through setibmopt(IBM\_IMAGE) or ioctl(SIOCSETRTTD).

When the application's socket is associated with more than one transport, every associated transport must support CSM buffers for a receive operation to be accepted. For a send operation, the transport chosen by the system for the destination IP address must support CSM buffers.

## Related services

[“recvmsg \(BPX2RMS, BPX4RMS\) — Receive messages on a socket and store them in message buffers”](#) on page 687

## Characteristics and restrictions

None.

### Examples

See “BPX1SRX (srx\_np) example” on page 1296.

## stat (BPX1STA, BPX4STA) – Get status information about a file by path name

---

### Function

The stat callable service obtains status information about a specified file. You specify the file by its path name.

If the path name that is specified refers to a symbolic link, the symbolic link name is resolved to a file, and the status information for that file is returned. To obtain status information about a symbolic link, rather than a file it refers to, use “lstat (BPX1LST, BPX4LST) – Get status information about a file or symbolic link by path name” on page 409.

For the corresponding service using a file descriptor, see “fstat (BPX1FST, BPX4FST) – Get status information about a file by descriptor ” on page 238.

### Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1STA)</b>	31-bit
<b>AMODE (BPX4STA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1STA, (Pathname_length,
               Pathname,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4STA with the same parameters.

### Parameters

#### Pathname\_length

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of Pathname.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of an area, of length Pathname\_length, that contains the path name of the file for which you want to obtain status. The Pathname can be a path name to a file, a link named by a path name to a file (as created by [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file ” on page 383](#)), or a symbolic link named by a path name to a file (as created by [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)).

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Status\_area\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the area to which the service returns Status\_area. To determine the value of Status\_area\_length, use the BPXYSTAT macro (see [“BPXYSTAT — Map the response structure for stat ” on page 1162](#)).

**Status\_area**

Parameter supplied and returned

**Type:**

Structure

**Length:**

Specified by the Status\_area\_length parameter

The name of an area, of length Status\_area\_length, to which the service returns the status information for the file. The Status\_area is mapped by the BPXYSTAT macro (see [“BPXYSTAT — Map the response structure for stat ” on page 1162](#)).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the stat service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the stat service stores the return code. The stat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The stat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The process does not have permission to search some component of the Pathname prefix.
<b>EINVAL</b>	Parameter error—for example, a zero-length buffer. The following reason code can accompany the return code: JRBuffTooSmall.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or some component of the pathname is longer than 255 characters. This could be as a result of encountering a symbolic link during resolution of Pathname, if the substituted string is longer than 1023 characters.
<b>ENOENT</b>	No file named Pathname was found, or Pathname was not specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	A component of the Pathname prefix is not a directory.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the stat service stores the reason code. The stat service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. All time fields in the Status\_area are in POSIX format.
2. The File Mode field in the Status\_area is mapped by the BPXYMODE macro (see “[BPXYMODE — Map the mode constants](#)” on page 1106). For information about the values for file type, see “[BPXYFTYP — File type definitions](#)” on page 1078.
3. If no security label (SECLABEL) exists for the file, the security label field in the Status\_area contains binary zeros.

## Related services

- [“chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory”](#) on page 101

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“fpathconf \(BPX1FPC, BPX4FPC\) — Determine configurable path name variables” on page 233](#)
- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#)
- [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor ” on page 241](#)
- [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file ” on page 383](#)
- [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor ” on page 387](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe” on page 551](#)
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket” on page 652](#)
- [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)
- [“utime \(BPX1UTI, BPX4UTI\) — Set file access and modification times ” on page 981](#)
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket” on page 1032](#)

## Characteristics and restrictions

To obtain information about a file, you need not have permissions for the file itself. However, you must have search permission for all the directory components of Pathname.

## Examples

See [“BPX1STA \(stat\) example” on page 1297](#).

# statvfs (BPX1STV, BPX4STV) — Get the file system status

---

## Function

The statvfs callable service obtains status information about a file system that is specified by a file pathname from the desired file system.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1STV):</b>	31-bit
<b>AMODE (BPX4STV):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1STV, (Pathname_length,
               Pathname,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4STV with the same parameters.

**Parameters****Pathname\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the pathname.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

Printable characters

**Length:**

Pathname\_length

The name of a field, of length Pathname\_length, that specifies a file pathname in the file system about which status is desired.

**Status\_area\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the length of the area to which the service returns status information.

**Status\_area**

Parameter supplied and returned

**Type:**

Structure

**Length:**

Specified by the Status\_area\_length parameter

The name of an area of length `Status_area_length` to which the service returns the status information for the file system. The `BPXYSSTF` macro maps this area. For information on this macro, see [“BPXYSSTF — Map response structure for file system status”](#) on page 1161.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the `statvfs` service returns the length of the status written to the `Status_area` if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the `statvfs` service stores the return code. The `statvfs` service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The `statvfs` service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	The calling process does not have permission to search some component of the Pathname prefix.
<b>EAGAIN</b>	Information is temporarily unavailable. This can occur if the mount process for the file system is not complete.
<b>EINVAL</b>	Parameter error; for example, <code>Status_area_length</code> is too small. The following reason code can accompany the return code: <code>JRBufTooSmall</code> .
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters.
<b>ENOENT</b>	No file named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: <code>JRFileNotThere</code> .
<b>ENOTDIR</b>	Some component of the Pathname prefix is not a directory.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the `statvfs` service stores the reason code. The `statvfs` service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



## Usage notes

1. Provided that the passed Status\_area\_length is not less than or equal to zero, it is not considered an error if the Status\_area\_length is not sufficient to hold all the requested information. (That is, future expansion is allowed for.) As much information as will fit is written to Status\_area, and this amount is returned.
2. The amount of valid data that is returned in the Status\_area is indicated by the Return\_value. This allows for differences in the release levels of z/OS UNIX and the physical file systems.

## Related services

- [“fstatvfs \(BPX1FTV, BPX4FTV\) — Get the file system status” on page 244](#)
- [“w\\_statvfs \(BPX1STF, BPX4STF\) — Get the file system status ” on page 1029](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1STV \(statvfs\) example” on page 1299](#).

# sw\_sigdlv (BPX1DSD, BPX4DSD) – Switch the setting for signal delivery

---

## Function

The sw\_sigdlv callable service enables or disables signal delivery for the current process.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1DSD):</b>	31-bit
<b>AMODE (BPX4DSD):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1DSD,(signal_ind)
```

AMODE 64 callers use BPX4DSD with the same parameter.

## Parameters

### signal\_ind

Supplied parameter

### Type

Integer

### Length

Fullword

The name of a fullword containing a numeric value that indicates whether signal delivery should be enabled or disabled. The `signal_ind` constants are defined in the `BPXYCONS` macro. See [“BPXYCONS — Constants used by services”](#) on page 1062.

Constant	Description
<b>SW_SIGDLV_ENABLE#</b>	Enable signal delivery
<b>SW_SIGDLV_DISABLE#</b>	Disable signal delivery

## Usage notes

There are no returns from the `sw_sigdlv` callable service. The task is abended if an error occurs.

## Related services

- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ”](#) on page 360
- [“mvsunsigsetup \(BPX1MSD, BPX4MSD\) — Detach the signal setup ”](#) on page 495
- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals”](#) on page 487
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ”](#) on page 845

## Characteristics and restrictions

None.

# symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name

---

## Function

The `symlink` callable service creates a symbolic link to a path name. A file of type symbolic link is created.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SYM)</b>	31-bit
<b>AMODE (BPX4SYM)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1SYM, (Pathname_length,
               Pathname,
               Link_name_length,
               Link_name,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SYM with the same parameters.

**Parameters****Pathname\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the length of Pathname.

**Pathname**

Supplied parameter.

**Type**

Character string.

**Character set:**

No restriction.

**Length**

Specified by the Pathname\_length parameter.

The name of a field, of length Pathname\_length, that contains the path name for which you are creating a symbolic link.

Path names can begin with or without a slash:

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name, and the search for the file starts at the parent directory of the symbolic link file.

**Link\_name\_length**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the length of Link\_name. The name can be up to 1023 bytes long; each component of the name (between delimiters) can be up to 255 bytes long.

**Link\_name**

Supplied parameter.

**Type**

Character string.

**Character set**

No restriction.

**Length**

Specified by the Link\_name\_length parameter.

The name of a field that contains the symbolic link that is being created.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the symlink service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the symlink service stores the return code. The symlink service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The symlink service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The requested operation requires writing in a directory with a mode that denies write permission.
<b>EEXIST</b>	The link name already exists. The following reason code can accompany the return code: JRSymFileAlreadyExists.
<b>EFBIG</b>	A request to create a symbolic link is prohibited because the file size limit for the process is set to 0.
<b>EINVAL</b>	This error code might be returned for any of the following reasons: <ul style="list-style-type: none"> <li>• A component of the path prefix of Pathname or the entire path name exceeds the maximum allowed.</li> <li>• The value of Pathname_length is less than or equal to zero.</li> <li>• A null character appears in Pathname.</li> </ul> The following reason code can accompany the return code: JRCompNotDir, JRInvalidSymLinkCom, JRInvalidSymLinkLen, and JRNullInPath.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
<b>ENAMETOOLONG</b>	Pathname or Link_name is longer than 1023 characters, or some component of that name is longer than 255 characters. Name truncation is not supported.

Return_code	Explanation
<b>ENOENT</b>	Link_name has a slash as its last component, which indicates that the preceding component is a directory. A symbolic link cannot be a directory.
<b>ENOSPC</b>	The directory in which the entry for the symbolic link is being placed cannot be extended; not enough space remains in the file system.
<b>ENOTDIR</b>	A component of the path prefix of Link_name is not a directory.
<b>EROFS</b>	The requested operation requires writing in a directory on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the symlink service stores the reason code. The symlink service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The symlink service creates a symbolic link (Link\_name) with the file that you specify by Pathname.
2. Like a hard link (described in [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file”](#) on page 383), a symbolic link allows a file to have more than one name. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link, however, provides no such assurance; in fact, the file identified by Pathname need not exist when the symbolic link is created.
3. When a component of a path name refers to a symbolic link rather than to a directory, the path name that is contained in the symbolic link is resolved. If the path name in the symbolic link begins with / (slash), the symbolic link path name is resolved relative to the process root directory. If the path name in the symbolic link does not begin with /, the symbolic link path name is resolved relative to the directory that contains the symbolic link.
4. If the symbolic link is not the last component of the original path name, remaining components of the original path name are resolved from there.

When a symbolic link is the last component of a path name, it may or may not be resolved. Resolution depends on the function that is using the path name. For example, a rename request does not have a symbolic link resolved when it appears as the final component of the new or old path name. However, an open request does have a symbolic link resolved when it appears as the last component.

When a slash is the last component of a path name, and it is preceded by a symbolic link, the symbolic link is always resolved.

5. Because the mode of a symbolic link cannot be changed, its mode is ignored during the lookup process. Any files and directories to which a symbolic link refers are checked for access permission.
6. Sysplex members participating in a shared file system can access (read/write) file system data on other systems in the sysplex. For example, if SY1 and SY2 are two systems in a sysplex, a user on SY1 can access SY2's /etc directory.

The shared file system capability requires that /etc, /dev, /var, and /tmp be converted into symbolic links. If the content of the symbolic link begins with \$VERSION, \$SYSNAME, \$SYSSYMR, or \$SYSSYMA, the symbolic link will resolve in a specific manner:

- If the content of the symbolic link begins with \$SYSNAME and the BPXPRMxx parameter SYSPLEX is specified YES, \$SYSNAME is replaced with a slash followed by the system name (/SY1). If SYSPLEX(NO) is specified, \$SYSNAME is replaced with /SYSTEM. For example, if you have specified SYSPLEX(YES) and the symbolic link for /etc has the contents \$SYSNAME/etc, this will resolve to /SY1/etc on a system whose name is SY1, and will resolve to /SY2/etc on a system whose name is SY2.
- If the content of the symbolic link begins with \$VERSION, \$VERSION will resolve to the value specified on the VERSION parameter in BPXPRMxx. Thus, if VERSION in parmlib is set to REL9, resolution of a symbolic link with \$VERSION causes \$VERSION to be replaced with /REL9. For example, the symbolic link for /bin, which has the contents \$VERSION/bin, will resolve to /REL9/bin on a system whose \$VERSION value is set to REL9.
- When a component of the path name is a symbolic link that begins with \$SYSSYMR or \$SYSSYMA, any MVS static symbols in the template are replaced with the resolved substitution text. In the following examples, &SYSR is an MVS static symbol:

If the content of the symbolic link begins with \$SYSSYMR, \$SYSSYMR/ results in a relative path name; that is, the lookup proceeds from its current position in the path name. For example, if the symbolic link is /x/y/sym1 and the symbolic link contains \$SYSSYMR/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to OSV315/resdir. Because it is a relative path name (the identifier was \$SYSSYMR/), the resulting path name will be /x/y/OSV315/resdir.

If the content of the symbolic link begins with \$SYSSYMA, \$SYSSYMA/ results in an absolute path name; that is, the lookup starts over at the root. For example, if the symbolic link is /x/y/sym1 and the symbolic link contains \$SYSSYMA/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to /OSV315/resdir. Because it is an absolute path name (the identifier was \$SYSSYMA/), the resulting path name will be /OSV315/resdir.

Only the occurrence of \$SYSSYMR/ or \$SYSSYMA/ at the start will be recognized as an identifier for which the remaining text requires substitution. Any other identifiers after the beginning will remain as is in the resolved link name. There must be some text following a \$SYSSYMR/ or \$SYSSYMA/ for it to be recognized as a valid identifier with text containing symbols to be resolved.

If the content of the symbolic link begins with \$SYSSECA/ or \$SYSSECR/, the user's current security label is substituted into the path name. The symbol \$SYSSECA indicates that the user's current security label should be substituted into the path name as an absolute directory name. Pathname resolution continues at the ROOT with a directory name of the user's current security label. The symbol \$SYSSECR indicates that the user's current security label should be substituted into the path name as a relative directory. The path name resolution continues in the directory in which the symbolic link is encountered, with a directory name of the user's current security label. For more information about using security labels, see [Assigning security labels to data sets in z/OS Planning for Multilevel Security and the Common Criteria](#).

For more information about file system sharing in a sysplex, see in [Sharing file systems in a sysplex in z/OS UNIX System Services Planning](#). The SYSPLEX(YES|NO) and VERSION('nnnn') BPXPRMxx parameters are described in BPXPRMxx (z/OS UNIX System Services parameters) in [z/OS MVS Initialization and Tuning Reference](#).

7. Certain directories like /etc, /dev, /var, and /tmp are converted to symbolic links. Some shell commands have minor technical differences when referring to symbolic links than for regular files or directories. For example, ls does not follow symbolic links by default. /etc is a symbolic link, so ls /etc will display only the symbolic link name, in this case /etc.

In order to follow symbolic links, you must specify ls -L or provide a trailing slash. For example, ls -L /etc and ls /etc/ both display the files in the directory that the /etc symbolic link points to.

Other shell commands that have differences due to symbolic links are du, find, pax, rm and tar.

8. By default, the owning GID of the symbolic link is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:

- If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
- If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.

## Related services

- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“mknod \(BPX1MKN, BPX4MKN\) — Make a directory, a FIFO, a character special, or a regular file” on page 427](#)
- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link ” on page 666](#)
- [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory ” on page 690](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory ” on page 703](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1SYM \(symlink\) example” on page 1301](#).

# symlinkat (BPX1SYA, BPX4SYA) — Create a symbolic link relative to a directory file descriptor

---

## Function

The symlinkat service creates a symbolic link relative to a directory file descriptor.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SYA)</b>	31-bit
<b>AMODE (BPX4SYA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SYA,(OldPathname_length,
              OldPathname,
              NewPathName_length,
              NewPathname,
              NewDirfd,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SYA with the same parameters.

## Parameters

### OldPathname\_length

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

Length of the old path name.

### OldPathname

Supplied parameter.

#### Type

Character string.

#### Character set

No restriction.

The name of a field that contains the path name of the file.

### NewPathname\_length

Supplied parameter.

#### Type

Integer.

#### Character set

Fullword.

Length of the new path name.

### NewPathname

Supplied parameter.

#### Type

Character string.

#### Character set

No restriction.

#### Length

Specified by the Pathname\_length parameter.

The name of a field that contains the new path name of the file. If the path name given in *newpathname* is relative, then it is interpreted relative to the directory referred to by the file descriptor *newdirfd*. It is not interpreted relative to the current working directory of the calling process, as is done by symlink for a relative path name.

If *newpathname* is relative and *newdirfd* is the special value AT\_FDCWD, then *newpathname* is interpreted relative to the current working directory of the calling process (like symlink).

If *newpathname* is absolute, then *newdirfd* is ignored.



**NewDirfd**

Supplied parameter.

**Type**

Character string.

**Length**

Fullword.

The name of a fullword that contains the directory file descriptor of the file new directory file descriptor.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the symlinkat service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the symlinkat service stores the return code. The symlinkat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The symlink service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The requested operation requires writing in a directory with a mode that denies write permission.
<b>EBADF</b>	<i>newdirfd</i> is not a valid file descriptor.
<b>EEXIST</b>	The link name already exists.  The following reason codes can accompany the return code: JRSymFileAlreadyExists.
<b>EFBIG</b>	A request to create a symbolic link is prohibited because the file size limit for the process is set to 0.
<b>EINVAL</b>	This error code might be returned for any of the following reasons: <ul style="list-style-type: none"> <li>• A component of the path prefix of Pathname or the entire path name exceeds the allowed maximum.</li> <li>• The value of Pathname_length is less than or equal to zero.</li> <li>• A null character appears in Pathname.</li> </ul> The following reason codes can accompany the return code: JRCompNotDir, JRInvalidSymLinkCom, JRInvalidSymLinkLen, and JRNullInPath.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.

Return_code	Explanation
<b>ENAMETOOLONG</b>	Pathname or Link_name is longer than 1023 characters, or some component of that name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	Link_name has a slash as its last component, which indicates that the preceding component is a directory. A symbolic link cannot be a directory.
<b>ENOSPC</b>	The directory in which the entry for the symbolic link is being placed cannot be extended because not enough space remains in the file system.
<b>ENOTDIR</b>	A component of the path prefix of Link_name is not a directory.
<b>ENOTDIR</b>	<i>newpathname</i> is relative and <i>newdirfd</i> is a file descriptor that refers to a file other than a directory.
<b>EROFS</b>	The requested operation requires writing in a directory on a read-only file system. The following reason codes can accompany the return code: JRReadOnly.

### Reason\_code

Returned parameter.

### Type

Integer.

### Length

Fullword.

The name of a fullword in which the symlink service stores the reason code. The symlink service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## File system sharing in a sysplex

Sysplex members in a shared file system can access read/write file system data on other systems in the sysplex. For example, if SY1 and SY2 are two systems in a sysplex, a user on SY1 can access SY2's /etc directory.

The shared file system capability requires that /etc, /dev, /var, and /tmp be converted into symbolic links. If the content of the symbolic link begins with \$VERSION, \$SYSNAME, \$SYSSYMR, or \$SYSSYMA, the symbolic link are resolved in a specific manner.

- If the content of the symbolic link begins with \$SYSNAME and the BPXPRMxx parameter SYSPLEX is specified YES, \$SYSNAME is replaced with a slash followed by the system name (/SY1). If SYSPLEX(NO) is specified, \$SYSNAME is replaced with /SYSTEM. For example, if you specify SYSPLEX(YES) and the symbolic link for /etc has the contents \$SYSNAME/etc, this will resolve to /SY1/etc on a system whose name is SY1. On a system whose name is SY2, it resolves to /SY2/etc.
- If the content of the symbolic link begins with \$VERSION, \$VERSION will resolve to the value specified on the VERSION parameter in BPXPRMxx. Thus, if VERSION in parmlib is set to REL9, resolution of a symbolic link with \$VERSION causes \$VERSION to be replaced with /REL9. For example, the symbolic link for /bin, which has the contents \$VERSION/bin, will resolve to /REL9/bin on a system whose \$VERSION value is set to REL9.
- When a component of the path name is a symbolic link that begins with \$SYSSYMR or \$SYSSYMA, any static symbols in the template are replaced with the resolved substitution text. In the following examples, &SYSR is a static symbol.
  - If the content of the symbolic link begins with \$SYSSYMR, \$SYSSYMR/ results in a relative path name; that is, the lookup proceeds from its current position in the path name. For example, if the symbolic link is /x/y/sym1 and the symbolic link contains \$SYSSYMR/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to OSV315/resdir. Because

it is a relative path name (the identifier was \$SYSSYMR/), the resulting path name is /x/y/OSV315/resdir.

- If the content of the symbolic link begins with \$SYSSYMA, \$SYSSYMA/ results in an absolute path name; that is, the lookup starts over at the root. For example, if the symbolic link is /x/y/sym1 and the symbolic link contains \$SYSSYMA/&SYSR1./resdir, a path name lookup on /x/y/sym1 from SY1 will resolve the symbolic link to /OSV315/resdir. Because it is an absolute path name (the identifier was \$SYSSYMA/), the resulting path name is /OSV315/resdir.

Only the occurrence of \$SYSSYMR/ or \$SYSSYMA/ at the start is recognized as an identifier for which the remaining text requires substitution. Any other identifiers after the beginning remain unchanged in the resolved link name. Text must follow a \$SYSSYMR/ or \$SYSSYMA/ for it to be recognized as a valid identifier with text containing symbols to be resolved.

If the content of the symbolic link begins with \$SYSSECA/ or \$SYSSECR/, the user's current security label is substituted into the path name. The symbol \$SYSSECA indicates that the user's current security label should be substituted into the path name as an absolute directory name. Pathname resolution continues at the ROOT with a directory name of the user's current security label. The symbol \$SYSSECR indicates that the user's current security label should be substituted into the path name as a relative directory. The path name resolution continues in the directory in which the symbolic link is encountered, with a directory name of the user's current security label. For more information about using security labels, see [Assigning security labels to data sets](#) in *z/OS Planning for Multilevel Security and the Common Criteria*.

For more information about file system sharing in a sysplex, see in [Sharing file systems in a sysplex](#) in *z/OS UNIX System Services Planning*. The SYSPLEX(YES|NO) and VERSION('nnnn') BPXPRMxx parameters are described in [BPXPRMxx \(z/OS UNIX System Services parameters\)](#) in *z/OS MVS Initialization and Tuning Reference*.

## Usage notes

1. The symlink service creates a symbolic link (Link\_name) with the file that you specify by Pathname.
2. Like a hard link (described in “link (BPX1LNK, BPX4LNK) — Create a link to a file ” on page 383), a symbolic link allows a file to have more than one name. The presence of a hard link indicates the existence of a file, even after the original name is removed. However, a symbolic link provides no such assurance. In fact, the file that is identified by Pathname need not exist when the symbolic link is created.
3. When a component of a path name refers to a symbolic link rather than to a directory, the path name that is contained in the symbolic link is resolved. If the path name in the symbolic link begins with / (slash), the symbolic link path name is resolved relative to the process root directory. If the path name in the symbolic link does not begin with /, the symbolic link path name is resolved relative to the directory that contains the symbolic link.
4. If the symbolic link is not the last component of the original path name, remaining components of the original path name are resolved from there.

When a symbolic link is the last component of a path name, it might or might not be resolved. Resolution depends on the function that is using the path name. For example, a rename request does not have a symbolic link resolved when it appears as the final component of the new or old path name. However, an open request does have a symbolic link that is resolved when it appears as the last component.

When a slash is the last component of a path name, and it is preceded by a symbolic link, the symbolic link is always resolved.

5. Because the mode of a symbolic link cannot be changed, its mode is ignored during the lookup process. Any files and directories to which a symbolic link refers are checked for access permission.
6. Certain directories such as /etc, /dev, /var, and /tmp are converted to symbolic links.

Some shell commands have minor technical differences when they refer to symbolic links as opposed to regular files or directories. For example, ls does not follow symbolic links by default. /etc is a

symbolic link, so `ls /etc` displays only the symbolic link name. In this case, the symbolic link name is `/etc`.

In order to follow symbolic links, you must specify `ls -L` or provide a trailing slash. For example, `ls -L /etc` and `ls /etc/` both display the files in the directory that the `/etc` symbolic link points to.

Other shell commands that have differences due to symbolic links are **du**, **find**, **pax**, **rm**, and **tar**.

7. By default, the owning GID of the symbolic link is set to that of the parent directory. However, if the `FILE.GROUPOWNER.SETGID` profile exists in the `UNIXPRIV` class, the owning GID is determined by the `set-gid` bit of the parent directory.

- If the parent's `set-gid` bit is on, the owning GID is set to that of the parent directory.
- If the parent's `set-gid` bit is off, the owning GID is set to the effective GID of the process.

## Related services

- [“chown \(BPX1CHO, BPX4CHO\) — Change the owner or group of a file or directory” on page 104](#)
- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory ” on page 420](#)
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor ” on page 423](#)
- [“mknod \(BPX1MKN, BPX4MKN\) — Make a directory, a FIFO, a character special, or a regular file” on page 427](#)
- [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link by path name” on page 409](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link ” on page 666](#)
- [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory ” on page 690](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory ” on page 703](#)
- [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name ” on page 900](#)
- [“unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry ” on page 966](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1SYA \(symlinkat\) example” on page 1300](#) and [“BPX4SYA \(symlinkat\) example” on page 1392](#).

# sync (BPX1SYN, BPX4SYN) — Schedule file system updates

## Function

The `sync` callable service causes all information in memory that updates file systems to be scheduled for writing.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1SYN):</b>	31-bit

Operation	Environment
<b>AMODE (BPX4SYN):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SYN, (Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SYN.

## Parameters

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sync service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sync service stores the return code. The sync service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the sync service stores the reason code. The sync service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

The actual writing of data to all file systems is scheduled, but is not necessarily completed, upon return from the sync() service.

## Characteristics and restrictions

None.

## Examples

See “BPX1SYN (sync) example” on page 1301.

# syncfs (BPX1SYF, BPX4SYF) – Synchronizes the file system file referred to by the open file descriptor

---

## Function

The syncfs callable service synchronizes the file system file that is referred to by the open file descriptor *fd*.

For the macro, see “BPXYFCTL – Command values and flags for fcntl” on page 1076.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1SYF)</b>	31-bit
<b>AMODE (BPX4SYF)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1SYF,(File_descriptor,
              (Return_value,
               Return_code,
               Reason_code))
```

AMODE 64 callers use BPX4SYF.

## Parameters

### File\_descriptor

Supplied parameter.

### Type

File descriptor.

### Length

Fullword.

The name of a fullword that contains the file descriptor of the file whose attributes that you want to change.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the syncfs service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the syncfs service stores the return code. The syncfs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*

The syncfs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EBADF	<i>fd</i> is not a valid file descriptor.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the syncfs service stores the reason code. The syncfs service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

The actual writing of data to all file systems is scheduled, but is not necessarily completed, upon return from the syncfs service.

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1SYF \(syncfs\) example”](#) on page 1300 and [“BPX4SYF \(syncfs\) example”](#) on page 1393.

## sysconf (BPX1SYC, BPX4SYC) – Determine system configuration options

---

**Function**

The sysconf callable service gets the value of a configurable system variable.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SYC):	31-bit
AMODE (BPX4SYC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1SYC, (Sysconf_name,  
               Return_value,  
               Return_code,  
               Reason_code)
```

AMODE 64 callers use BPX4SYC with the same parameters.

Parameters

**Sysconf\_name**  
Supplied parameter

**Type:**  
Structure

**Length:**  
Fullword

The name of a fullword that specifies the configurable system variable that is to be retrieved. Each configurable system variable is mapped to a specific value that is defined in the BPXYCONS macro. See [“BPXYCONS — Constants used by services”](#) on page 1062.

Constant	Configurable system variable returned
SC_ARG_MAX	For ARG_MAX
SC_CHILD_MAX	For CHILD_MAX
SC_CLK_TCK	For CLK_TCK
SC_JOB_CONTROL	For _POSIX_JOB_CONTROL
SC_NGROUPS_MAX	For NGROUPS_MAX
SC_OPEN_MAX	For OPEN_MAX
SC_SAVED_IDS	For _POSIX_SAVED_IDS
SC_MMAP_MEM_MAX_NP	For MMAP_MEM_MAX_NP
SC_TTY_GROUP	For TTY GROUP
SC_THREADS_MAX_NP	For _THREADS_MAX_NP



**Constant****SC\_THREAD\_TASKS\_MAX\_NP****SC\_TZNAME\_MAX****SC\_VERSION****SC\_2\_CHAR\_TERM****SC\_PAGESIZE****SC\_PAGE\_SIZE****Configurable system variable returned**

For \_THREAD\_TASKS\_MAX\_NP

For TZNAME\_MAX

For \_POSIX\_VERSION

For CHAR\_TERM

For PAGESIZE

For PAGE\_SIZE

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sysconf service returns the actual value of the configurable system variable if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sysconf service stores the return code. The sysconf service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The sysconf service can return the following value in the Return\_code parameter:

**Return\_code****Explanation****EINVAL**

The value of the Sysconf\_name argument is not valid.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the sysconf service stores the reason code. The sysconf service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. SC\_MAX\_THREADS\_NP, SC\_CHILD\_MAX, SC\_OPEN\_MAX, SC\_MMAP\_MEM\_MAX, and SC\_MAX\_THREAD\_TASKS\_NP return the limits that are defined for the caller's process, not the system-wide limits.
2. SC\_PAGE\_SIZE and SC\_PAGESIZE return the page size based on the AMODE of the caller. AMODE 31 callers get a page size of 4 K, and AMODE 64 callers get a page size of 1M.

## Related services

- [“pathconf \(BPX1PCF, BPX4PCF\) — Determine configurable path name variables using a path name” on page 535](#)
- [“set\\_thread\\_limits \(BPX1STL, BPX4STL\) — Change task or thread limits for pthread\\_created threads” on page 795](#)

## Examples

See [“BPX1SYC \(sysconf\) example” on page 1300](#).

# takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program

---

## Function

The takesocket callable service acquires a specified socket from the program that is identified in the Clientid parameter.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1TAK)</b>	31-bit
<b>AMODE (BPX4TAK)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1TAK,(Clientid,
               Socket_Id,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TAK with the same parameters.

## Parameters

### Clientid

Supplied parameter

#### Type:

Structure

#### Length:

Length of BPXYCID

The name of a structure that contains Clientid information that identifies the (server) program from which the socket is to be taken. This information is typically obtained with the getclientid (BPX1GCL, BPX4GCL) service, issued by the server and passed to the subsidiary. See [“BPXYCID — Map the returning structure for getclientid\(\)” on page 1061](#) for more information about the format of this field. Clientid input may be as follows:

**CIdDomain**

Domain of the socket that is to be taken. See [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) for more information about the values that are defined for this field.

**CIdName**

One of the following:

- The server program's address space name.
- A fullword of binary zeros followed by the server program's process ID.

**CIdTask**

The server program's subtask identifier (supplied only if the address space name was supplied in the CIdName field).

**CIdReserved**

Binary zeros.

**Socket\_Id**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

An identifier for the socket that is being taken. This is supplied by the server program, and is either the socket descriptor obtained from an accept, or the socket token returned on a givesocket (BPX1GIV, BPX4GIV) service if givesocket was invoked with CIdType=CId#Close.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the takesocket service returns one of the following:

- -1 if the request is not successful.
- If not -1, the return value is the new socket descriptor.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the takesocket service stores the return code. The takesocket service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The takesocket service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The Socket_Id does not specify a valid socket that is owned by the other application; or the socket has already been taken.

Return_code	Explanation
<b>EACCES</b>	The other application did not give the socket to your application.
<b>EFAULT</b>	Using the Clientid parameter as specified would result in an attempt to access storage that is outside the caller's address space.
<b>EINVAL</b>	The Clientid parameter does not specify a valid client identifier: either the client's process cannot be found, or the client's process was found, but it has no outstanding givesockets.
<b>EMFILE</b>	The socket descriptor table is already full.
<b>EPERM</b>	The givesocket security label does not match the takesocket security label (JrUserNotAuthorized).

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the takesocket service stores the reason code. The takesocket service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The takesocket callable service used to be an MVS TCP/IP API, and was added to the z/OS UNIX callable services to allow migration of applications to a single library.
2. The Clientid output of getclientid (BPX1GCL, BPX4GCL) that is issued by the server program and passed to the subsidiary is intended to be used as the input Clientid of the takesocket service. This identifies the program from which the socket is to be taken. By using a FunctionCode of 2 on the getclientid service to obtain Clientid information that is to be used as the Clientid input of the takesocket service, the best performance of the takesocket service is achieved.

**Related services**

- [“getclientid \(BPX1GCL, BPX4GCL\) — Obtain the calling program's identifier”](#) on page 258
- [“givesocket \(BPX1GIV, BPX4GIV\) — Give a socket to another program ”](#) on page 332

**Characteristics and restrictions**

None.

## tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted

---

**Function**

The tcdrain callable service waits until all output sent to a file descriptor has been sent to the terminal device.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TDR):</b>	31-bit
<b>AMODE (BPX4TDR):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1TDR,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TDR with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor that represents the output device.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcdrain service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcdrain service stores the return code. The tcdrain service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in

*z/OS UNIX System Services Messages and Codes* The tcdrain service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBADF</b>	File_descriptor does not describe a valid open file.
<b>EINTR</b>	A signal interrupted the service before all output had been sent.
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENOTTY</b>	File_descriptor is not associated with a terminal.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcdrain service stores the reason code. The tcdrain service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. For secondary pseudoterminals, data is considered written when the manager pseudoterminal has read it.
2. The following table defines the processing of the SIGTTOU signal when tcdrain is called from a background process against a controlling terminal:

<b>SIGTTOU processing</b>	<b>Expected behavior</b>
<b>Default or signal handler</b>	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
<b>Ignored or blocked</b>	The SIGTTOU signal is not sent. The function continues normally.

**Related services**

- [“tcflow \(BPX1TFW, BPX4TFW\) — Suspend or resume data flow on a terminal ” on page 921](#)
- [“tcflush \(BPX1TFH, BPX4TFH\) — Flush input or output on a terminal ” on page 923](#)
- [“tcsendbreak \(BPX1TSB, BPX4TSB\) — Send a break condition to a terminal ” on page 934](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1TDR \(tcdrain\) example” on page 1302](#).

## tcflow (BPX1TFW, BPX4TFW) – Suspend or resume data flow on a terminal

---

### Function

The tcflow callable service suspends or resumes data flow on a terminal.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TFW):</b>	31-bit
<b>AMODE (BPX4TFW):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1TFW,(File_descriptor,
               Action,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TFW with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor for the terminal device.

#### Action

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains an indicator of the action that is to be taken. The possible constants are mapped in the BPXYTIOS macro (see [“BPXYTIOS — Map the termios structure”](#) on page 1169).

Constant	Description
<b>TCIOFF</b>	Send a STOP character to the terminal to stop the terminal from sending any further input.
<b>TCION</b>	Send a START character to the terminal to start the terminal sending input.
<b>TCOOFF</b>	Suspend output to the terminal.
<b>TCOON</b>	Resume output to the terminal.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcflow service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcflow service stores the return code. The tcflow service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcflow service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor does not describe a valid open file.
<b>EINTR</b>	A signal interrupted the call.
<b>EINVAL</b>	The Action parameter does not contain one of the expected values.
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENOTTY</b>	File_descriptor is not associated with a terminal.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcflow service stores the reason code. The tcflow service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.



## Usage notes

The following table defines the processing of the SIGTTOU signal when the tcflow service is called from a background process against a controlling terminal:

<b>SIGTTOU processing</b>	<b>Expected behavior</b>
<b>Default or signal handler</b>	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
<b>Ignored or blocked</b>	The SIGTTOU signal is not sent. The function continues normally.

## Related services

- [“tcdrain \(BPX1TDR, BPX4TDR\) — Wait until output has been transmitted ” on page 918](#)
- [“tcflush \(BPX1TFH, BPX4TFH\) — Flush input or output on a terminal ” on page 923](#)
- [“tcsendbreak \(BPX1TSB, BPX4TSB\) — Send a break condition to a terminal ” on page 934](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1TFW \(tcflow\) example” on page 1302](#).

# tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal

## Function

The tcflush callable service flushes all data that is sent to a device. Depending on the value of the Queue\_selector parameter, any data written, but not sent, or any data received, but not read, is discarded.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TFH):</b>	31-bit
<b>AMODE (BPX4TFH):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows: CALL BPX1TFH,(File\_descriptor, Queue\_selector, Return\_value, Return\_code, Reason\_code)

AMODE 64 callers use BPX4TFH with the same parameters.

**Parameters****File\_descriptor**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor of the terminal.

**Queue\_selector**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that specifies the queues that are to be flushed. The constants are mapped in the BPXYTIOS macro; see [“BPXYTIOS — Map the termios structure” on page 1169](#).

**Constant****Description****TCIFLUSH**

Flush data received but not read

**TCOFLUSH**

Flush data written but not sent

**TCIOFLUSH**

Flush both data received but not read and data written but not sent

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcflush service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcflush service stores the return code. The tcflush service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcflush service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is not a valid open file descriptor.
<b>EINTR</b>	A signal interrupted the call.
<b>EINVAL</b>	The Queue_selector specified was incorrect.
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENOTTY</b>	The file that is associated with the file descriptor is not a terminal.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcflush service stores the reason code. The tcflush service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

The following table defines the processing of the SIGTTOU signal when tcflush is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
<b>Default or signal handler</b>	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
<b>Ignored or blocked</b>	The SIGTTOU signal is not sent. The function continues normally.

**Related services**

- [“tcdrain \(BPX1TDR, BPX4TDR\) — Wait until output has been transmitted ” on page 918](#)
- [“tcflow \(BPX1TFW, BPX4TFW\) — Suspend or resume data flow on a terminal ” on page 921](#)
- [“tcsendbreak \(BPX1TSB, BPX4TSB\) — Send a break condition to a terminal ” on page 934](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1TFH \(tcflush\) example” on page 1302](#).

## tcgetattr (BPX1TGA, BPX4TGA) – Get the attributes for a terminal

### Function

The tcgetattr callable service gets control information for a terminal and stores it in the specified Termios\_structure.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, state any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TGA):</b>	31-bit
<b>AMODE (BPX4TGA):</b>	64-bit
<b>ASC mode:</b>	Primary address space control (ASC) mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1TGA,(File_descriptor,
              Termios_structure,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGA with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor of the terminal for which you want attributes.

#### Termios\_structure

Returned parameter

##### Type:

Structure

##### Length:

Specified by BPXYTIOS#LENGTH in the BPXYTIOS macro

The name of an area into which the function is to return the terminal information. Termios\_structure is mapped by the BPXYTIOS macro. This structure contains the control modes, input modes,

output modes, local modes, and special control characters as defined by the POSIX standard (see [“BPXYTIOS — Map the termios structure”](#) on page 1169).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcgetattr service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcgetattr service stores the return code. The tcgetattr service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcgetattr service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is not a valid open file descriptor.
<b>ENOTTY</b>	The file that is associated with the file descriptor is not a terminal; the process does not have a controlling terminal; or the file is not the controlling terminal for the process.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcgetattr service stores the reason code. The tcgetattr service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The BPXYTIOS macro should be used to map the termios structure and define the equates for bits and values. Note the following about BPXYTIOS:
  - BPXYTIOS generates standard POSIX-defined names, except that all names are uppercase. In addition, all names can have a user-specified prefix.
  - When testing or setting bits in flag fields, you should use an offset name to define which byte in the flag field contains the bit. For instance: TM C\_CFLAG+HUPCL\_O,HUPCL.
  - CS5 through CS8 values can be contained in CSIZE. CSIZE is essentially a 2-bit integer that can contain decimal values 0 through 3, as defined by CS5 through CS8.
  - BPXYTIOS can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.

## tcgetcp (BPX1TGC, BPX4TGC)

- The C\_CC field is an array of 1-byte fields, indexed by the various special character equates. These equates can be used as offsets into C\_CC, or can be put into a register to be used with indexing instructions. For instance:

MVC	C_CC+VSUSP,NEWVAL	To set a new value
LA	R10,VSUSP	To set an register to use as an index in a later IC or STC instructions

2. You can run the tcgetattr service in either a foreground or a background process. However, if the process is in the background, a foreground process can later change the attributes that you obtained.

### Related services

[“tcsetattr \(BPX1TSA, BPX4TSA\) — Set the attributes for a terminal ” on page 937](#)

### Characteristics and restrictions

None.

### Examples

See [“BPX1TGA \(tcgetattr\) example” on page 1302.](#)

## tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names

### Function

The tcgetcp callable service gets the terminal session code page names and Code Page Change Notification (CPCN) capability.

### Requirements

#### Operation

#### Authorization:

#### Dispatchable unit mode:

#### Cross memory mode:

#### AMODE (BPX1TGC):

#### AMODE (BPX4TGC):

#### ASC mode:

#### Interrupt status:

#### Locks:

#### Control parameters:

#### Environment

Supervisor or problem state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1TGC,(File_descriptor,  
              Termcp_length,  
              Termcp_structure,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4TGC with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor of the terminal for which you want to get the code page names and data conversion environment.

### Termcp\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of the fullword that contains the length of the Termcp\_structure. The Termcp\_structure is mapped by BPXYTCCP, and has a length of TCCP#LENGTH. See [“BPXYTCCP — Map the terminal control code page structure” on page 1163](#).

### Termcp\_structure

Returned parameter

#### Type:

Structure

#### Length:

Specified by Termcp\_length.

The name of an area where the tcgetcp service returns the Termcp\_structure. The Termcp\_structure is mapped by the BPXYTCCP macro. See [“BPXYTCCP — Map the terminal control code page structure” on page 1163](#).

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcgetcp service returns one of the following:

- 1, if the terminal device supports a capability of forward code page names only.
- 2, if the terminal device supports a capability of forward code page names and tables.
- -1, if the request is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the tcgetcp service stores the return code. The tcgetcp service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes](#)

(*errno*s) in *z/OS UNIX System Services Messages and Codes* The tcgetcp service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is an incorrect open file descriptor.
<b>EINVAL</b>	One of the parameters contains a value that is not correct. Consult Reason_Code returned to determine the exact reason the error occurred.
<b>ENODEV</b>	One of the following error conditions exists: <ul style="list-style-type: none"> <li>• The terminal device driver does not support CPCN functions.</li> <li>• CPCN functions have not been enabled. For a pseudoterminal device file, issue the tcsetcp (BPX1TSC, BPX4TSC) callable service against the manager pseudoterminal first to enable CPCN support.</li> </ul>
<b>ENOTTY</b>	The file that is associated with the file descriptor is not a terminal device.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetcp service stores the reason code. The tcgetcp service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. For terminal devices that support forward code page names only CPCN capability, use the tcsetcp (BPX1TSC, BPX4TSC) callable service to change the terminal session data conversion environment.

The pseudoterminal device driver supports this CPCN capability.

2. For terminal devices that support forward code page names and tables CPCN capability, use the tcsettables (BPX1TST, BPX4TST) callable service to change the terminal session code conversion environment.

The OCS remote-tty device driver supports this CPCN capability.

3. In the returned Termcp\_structure, if the TCCPBINARY flag is set, the code page names should not be used. BINARY indicates that the data conversion point is to perform no data conversion for the terminal session.
4. For pseudoterminal support, the tcsetcp (BPX1TSC, BPX4TSC) callable service must be against the manager pseudoterminal for CPCN functions to be enabled.
5. In the returned Termcp\_structure, if the TCCPFASTP flag is set, the data conversion that is specified by the source and target code page names can be performed locally to the data conversion application. This is valid any time that a table-driven conversion can be performed. For example, the data conversion point (application) could use the z/OS UNIX iconv() service to build local data conversion tables and perform all data conversion using the local tables, instead of using iconv() all in subsequent conversions. This provides for better-performing data conversion.
6. The BPXYTCCP macro should be used to map the Termcp\_structure and define the equates for the flag byte values. Note the following about BPXYTCCP:
  - BPXYTCCP can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
  - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME should be terminated by a NUL (X'00') character.



- The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME are case sensitive.

## Related services

- [“tcsetcp \(BPX1TSC, BPX4TSC\) — Set terminal code page names ” on page 940](#)
- [“tcsettables \(BPX1TST, BPX4TST\) — Set terminal code page names and conversion tables” on page 946](#)

## Characteristics and restrictions

The tcgetcp service is supported by the pseudoterminal and (OCS) remote terminal device drivers.

## Examples

See [“BPX1TGC \(tcgetcp\) example” on page 1303](#).

# tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID

---

## Function

The tcgetpgrp callable service gets the process group ID of the foreground process group that is associated with a terminal, which is identified by its file descriptor.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TGP):</b>	31-bit
<b>AMODE (BPX4TGP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1TGP,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGP with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor for the terminal.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetpgrp service returns the process group ID of the foreground process group that is associated with the terminal, if the request is successful; or -1, if it is not successful. If there is no foreground process group, a positive value, not equal to any existing process group, is returned.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetpgrp service stores the return code. The tcgetpgrp service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcgetpgrp service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBADF</b>	The File_descriptor parameter does not specify a valid open file descriptor.
<b>ENOTTY</b>	The file descriptor is not associated with a terminal.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetpgrp service stores the reason code. The tcgetpgrp service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Related services**

- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ”](#) on page 777
- [“setsid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ”](#) on page 793
- [“tcsetpgrp \(BPX1TSP, BPX4TSP\) — Set the foreground process group ID ”](#) on page 944

**Characteristics and restrictions**

None.

### Examples

See “BPX1TGP (tcgetpgrp) example” on page 1303.

## tcgetsid (BPX1TGS, BPX4TGS) — Get a process group ID for the session leader for the controlling terminal

### Function

The tcgetsid callable service obtains the process group ID of the session leader that is associated with the terminal that is specified by the file descriptor.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TGS):</b>	31-bit
<b>AMODE (BPX4TGS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1TGS,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGS with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor for the terminal.

#### Return\_value

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetsid service returns the process group ID associated with the terminal if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetsid service stores the return code. The tcgetsid service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcgetsid service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The File_descriptor parameter is not associated with a controlling terminal.
<b>EBADF</b>	The File_descriptor parameter does not specify a valid file descriptor.
<b>ENOTTY</b>	The file descriptor is not associated with a terminal device.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcgetsid service stores the reason code. The tcgetsid service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Characteristics and restrictions**

None.

## **tcsendbreak (BPX1TSB, BPX4TSB) – Send a break condition to a terminal**

---

**Function**

The tcsendbreak callable service sends a BREAK signal to a terminal that uses asynchronous serial data transmission.

If the target terminal is an OCS-attached serial terminal, the BREAK signal is sent to the terminal. If the target terminal is a pseudoterminal, control returns without any significant action.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1TSB):	31-bit
AMODE (BPX4TSB):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

The syntax format is as follows:

```
CALL BPX1TSB,(File_descriptor,
              Duration,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TSB with the same parameters.

Parameters

File\_descriptor

Supplied parameter

Type:

Integer

Length:

Fullword

The name of a fullword that contains the file descriptor for the terminal device to which the break is to be sent.

Duration

Supplied parameter

Type:

Integer

Length:

Fullword

The name of a fullword that contains the duration of the BREAK transmission. If the target terminal is a pseudoterminal, the Duration parameter has no effect.

Return\_value

Returned parameter

Type:

Integer

Length:

Fullword

The name of a fullword in which the tcsendbreak service returns 0 if the request is successful, or -1 if it is not successful.

Return\_code

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsendbreak service stores the return code. The tcsendbreak service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcsendbreak service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is not a valid open file descriptor.
<b>EINTR</b>	The tcsendbreak (BPX1TSB, BPX4TSB) service was called from a background job, and the SIGTTOU signal either had default action or a signal handler. The function was not performed.
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENOTTY</b>	File_descriptor is not associated with a terminal.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsendbreak service stores the reason code. The tcsendbreak service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

The following table defines the processing of the SIGTTOU signal when tcsendbreak is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
<b>Default or signal handler</b>	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
<b>Ignored or blocked</b>	The SIGTTOU signal is not sent. The function continues normally.

**Related services**

- [“tcdrain \(BPX1TDR, BPX4TDR\) — Wait until output has been transmitted ”](#) on page 918
- [“tcflow \(BPX1TFW, BPX4TFW\) — Suspend or resume data flow on a terminal ”](#) on page 921
- [“tcflush \(BPX1TFH, BPX4TFH\) — Flush input or output on a terminal ”](#) on page 923

## Characteristics and restrictions

None.

## Examples

See “BPX1TSB (tcsendbreak) example” on page 1305.

# tcsetattr (BPX1TSA, BPX4TSA) – Set the attributes for a terminal

## Function

The tcsetattr callable service sets control information for a terminal from the specified Termios\_structure.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TSA):</b>	31-bit
<b>AMODE (BPX4TSA):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1TSA,(File_descriptor,
              Actions,
              Termios_structure,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TSA with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor of the terminal for which attributes are to be set.

### Actions

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains indicators that control the setting of the attributes. The following possible values are defined in the BPXYTIOS macro; see [“BPXYTIOS — Map the termios structure”](#) on page 1169.

**Constant****Description****TCSANOW**

Change the terminal attributes immediately.

**TCSADRAIN**

Change the terminal attributes when all output to the terminal has been sent.

**TCSAFLUSH**

Change the terminal attributes when all output to the terminal has been sent; and all input that has been received, but not read, is to be discarded.

**Termios\_structure**

Supplied parameter

**Type:**

Structure

**Length:**

Specified by BPXYTIOS#LENGTH in the BPXYTIOS macro

The name of an area that contains the attributes that are to be set. Termios\_structure is mapped by the BPXYTIOS macro. This structure contains the control modes, input modes, output modes, local modes, and special control characters. For the layout of the Termios\_structure, see [“BPXYTIOS — Map the termios structure”](#) on page 1169.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetattr service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetattr service stores the return code. The tcsetattr service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcsetattr service can return one of the following values in the Return\_code parameter:

**Return\_code****Explanation****EBADF**

File\_descriptor is an incorrect open file descriptor.

**EINTR**

A signal interrupted the call.

**EINVAL**

An action or value that was specified was incorrect.



Return_code	Explanation
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENOTTY</b>	The file that is associated with the file descriptor is not a terminal.
<b>EPERM</b>	A change was made that is not permitted from a subsidiary pseudoterminal. See <a href="#">“Characteristics and restrictions” on page 940</a> .

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetattr service stores the reason code. The tcsetattr service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. A program should always issue the tcsetattr callable service using a termios structure that was returned from a previous call to tcgetattr (BPX1TGA, BPX4TGA) (see [“tcgetattr \(BPX1TGA, BPX4TGA\) – Get the attributes for a terminal” on page 926](#)), with appropriate changes to the various fields.
2. The BPXYTIOS macro should be used to map the termios structure and define the equates for bits and values. Note the following about BPXYTIOS:
  - BPXYTIOS generates standard POSIX-defined names, except that all names are uppercase. In addition, all names can have a user-specified prefix.
  - When testing or setting bits in flag fields, you should use an offset name to define which byte in the flag field contains the bit. For instance: TM\_C\_CFLAG+HUPCL\_O,HUPCL.
  - CS5 through CS8 values can be contained in CSIZE. CSIZE is essentially a 2-bit integer that can contain decimal values 0 through 3, as defined by CS5 through CS8.
  - BPXYTIOS can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
  - The C\_CC field is an array of 1-byte fields, indexed by the various special character equates. These equates can be used as offsets into C\_CC, or can be put into a register to be used with indexing instructions. For instance:

```

MVC  C_CC+VSUSP,NEWVAL    To set a new value
LA   R10,VSUSP           To set a register to use as an index
                           in a later IC or STC instruction

```

3. The following table defines the processing of the SIGTTOU signal when the tcsetattr (BPX1TSA, BPX4TSA) service is called from a background process against a controlling terminal:

<b>SIGTTOU processing</b>	<b>Expected behavior</b>
<b>Default or signal handler</b>	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
<b>Ignored or blocked</b>	The SIGTTOU signal is not sent. The function continues normally.

**Related services**

- [“tcgetattr \(BPX1TGA, BPX4TGA\) — Get the attributes for a terminal ” on page 926](#)

**Characteristics and restrictions**

- The subsidiary pseudoterminal cannot set the PACKET, PKTXTND, or PKT3270 bits.
- Neither the subsidiary pseudoterminal nor the manager pseudoterminal can set the PTU3270 bit if PKT3270 is not on.
- The manager pseudoterminal cannot set the PKT3270 bit unless PKRXTND is also on.

**Examples**

See [“BPX1TSA \(tcsetattr\) example” on page 1304.](#)

## **tcsetcp (BPX1TSC, BPX4TSC) — Set terminal code page names**

---

**Function**

The tcsetcp callable service sets the terminal session code page names to the specified values.

**Requirements**

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TSC):</b>	31-bit
<b>AMODE (BPX4TSC):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1TSC,(File_descriptor,
               Termcp_length,
               Termcp_structure,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TSC with the same parameters.

**Parameters****File\_descriptor**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor of the terminal for which the code page names are to be set.

**Termcp\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of the fullword that contains the length of the Termcp\_structure. The Termcp\_structure is mapped by BPXYTCCP, and has a length of TCCP#LENGTH. See [“BPXYTCCP – Map the terminal control code page structure”](#) on page 1163.

**Termcp\_structure**

Supplied parameter

**Type:**

Structure

**Length:**

Specified by Termcp\_length

The name of an area that contains the code page information to be set. This structure contains the source (ASCII) code page name, target (EBCDIC) code page name, and control flags. The Termcp\_structure is mapped by the BPXYTCCP macro (see [“BPXYTCCP – Map the terminal control code page structure”](#) on page 1163).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetcp service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetcp service stores the return code. The tcsetcp service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcsetcp service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is an incorrect open file descriptor.
<b>EINTR</b>	A signal interrupted the call.
<b>EINVAL</b>	One of the parameters contains a value that is not correct. Consult Reason_Code returned to determine the exact reason the error occurred.

Return_code	Explanation
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENODEV</b>	One of the following error conditions exists: <ul style="list-style-type: none"> <li>• CPCN functions have not been enabled. tcsetcp must be issued against the manager pseudoterminal before any CPCN function can be issued against the subsidiary pseudoterminal.</li> <li>• The terminal device driver does not support the forward code page names only CPCN capability.</li> </ul>
<b>ENOTTY</b>	The file that is associated with the file descriptor is not a terminal device.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetcp service stores the reason code. The tcsetcp service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

**Attention:** Use this service carefully. By changing the code pages for the data conversion, you may cause unpredictable behavior in the terminal session if the actual data used for the session is not encoded to the specified source (ASCII) and target (EBCDIC) code pages.

1. Use the tcsetcp callable service to send new code page names to the terminal session data conversion point to change the data conversion environment.

The tcsetcp callable service is used with terminal devices that support the forward code page names only CPCN capability. Use the tcgetcp (BPX1TGC, BPX4TGC) callable service to determine the terminal device CPCN capability.

2. The BPXYTCCP macro should be used to map the Termcp\_structure and define the equates for the flag byte values. Note the following about BPXYTCCP:

- BPXYTCCP can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
- The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME must be terminated by a NUL (X'00') character.
- The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME are case-sensitive.

3. The tcsetcp callable service is supported by the pseudoterminal (pty) device driver. For terminal sessions that use pty support, the data conversion point is the application that uses the manager pseudoterminal. An example data conversion point is the rlogin server. Here, rlogin uses CPCN functions to change the ASCII source or EBCDIC target code pages to use in its data conversion for the terminal session.

During its processing of the tcsetcp service, the pty device driver applies the new code page names once the pty outbound data queue is drained. When this occurs, the pty input data queue is also flushed, and a TIOCPKT\_CHCP packet exception event is generated (if extended packet mode is enabled) to notify the manager pseudoterminal application that the code page names have been changed. The manager pseudoterminal application can then use the tcgetcp (BPX1TGC, BPX4TGC)

callable service to retrieve the new code page names and establish the new data conversion environment.

The tcsetcp service is supported by both the manager and subsidiary pseudoterminal device drivers. However, CPCN functions must first be enabled by the application that uses the manager pseudoterminal; enabling CPCN functions is performed by the system during the initial tcsetcp invocation against the manager pseudoterminal device. When the tcsetcp invocation is performed against the manager pseudoterminal it may be subsequently issued against the subsidiary pseudoterminal.

4. The data conversion for a terminal session is performed on a session (terminal file) basis. If you change the data conversion characteristics for one file descriptor, the new data conversion applies to all open file descriptors that are associated with this terminal file.
5. Use the tcsetcp callable service to notify the data conversion point to stop data conversion. This is done by setting the TCCPBINARY flag. If this flag is set, the source and target code page names (TCCPSRCNAME and TCCPTRGNAME, respectively) are not changed from their current values.



**Attention:** Use this option carefully. When the data conversion is disabled, the z/OS shell cannot be used until the data conversion is reenabled, using valid code pages for the terminal session.

6. Use the TCCPFASTP flag to indicate to the data conversion point (such as rlogin) that the data conversion that is specified by the source and target code page names can be performed locally to the application. This is valid anytime that a table-driven conversion can be performed. For example, the data conversion point (application) could use iconv() to build the local data conversion tables and perform all data conversion using the local tables, instead of using iconv() in subsequent conversions. This provides for better-performing data conversion.
7. The following table defines the processing of the SIGTTOU signal when the tcsetcp service is called from a background process group against its controlling terminal:

#### **SIGTTOU processing**

#### **Expected behavior**

##### **Default or signal handler**

The SIGTTOU signal is generated.  
The function is not performed.  
Return\_value is set to -1,  
and Return\_code is set to EINTR.

##### **Ignored or blocked**

The SIGTTOU signal is not sent.  
The function continues normally.

## **Related services**

- [“tcgetcp \(BPX1TGC, BPX4TGC\) — Get terminal code page names ” on page 928](#)
- [“tcsettables \(BPX1TST, BPX4TST\) — Set terminal code page names and conversion tables” on page 946](#)

## **Characteristics and restrictions**

The tcsetcp service is supported by the pseudoterminal device driver.

## **Examples**

See [“BPX1TSC \(tcsetcp\) example” on page 1305](#).

## tcsetpgrp (BPX1TSP, BPX4TSP) – Set the foreground process group ID

---

### Function

The tcsetpgrp callable service moves the requested process group into the foreground, replacing the current foreground process group. The current foreground process group then becomes a background process group.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TSP):</b>	31-bit
<b>AMODE (BPX4TSP):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

```
CALL BPX1TSP,(File_descriptor,
               Process_group_id,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TSP with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor of the terminal device.

#### Process\_group\_ID

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the process group ID that is to be associated with the controlling terminal.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetpgrp service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetpgrp service stores the return code. The tcsetpgrp service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcsetpgrp service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is not a valid open file descriptor.
<b>EINTR</b>	A signal interrupted the function.
<b>EINVAL</b>	Process_group_ID is not a process group ID that is supported by this implementation.
<b>ENOTTY</b>	The calling process does not have a controlling terminal; File_descriptor is not associated with the controlling terminal; or the controlling terminal is no longer associated with the session of the calling process.
<b>EPERM</b>	Process_group_ID does not match the process group ID of any process in the same session as the calling process.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsetpgrp service stores the reason code. The tcsetpgrp service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The tcsetpgrp service moves the requested process group into the foreground, replacing the current foreground process group. The current foreground process group then becomes a background group. This terminal must be the controlling terminal of the calling process, and it must be currently associated with the session of the calling process. Process\_group\_ID must represent a process group in the same session as the calling process.
2. After the foreground process group is set, reads by the process group that was formerly in the foreground fail or cause the process group to stop from a SIGTTIN signal. Writes can also cause the process to stop (from a SIGTTOU signal), or they can succeed, depending upon the current setting of TOSTOP (from tcsetattr) and the signal options for SIGTTOU.

3. The system issues a SIGTTOU signal when tcsetpgrp() is issued from a background process, unless SIGTTOU is being ignored or blocked. If the signal is set to default processing (SIG\_DFL), the process group is stopped. If there is a handler, the handler gets control and errno=EINTR is returned.
4. The File\_descriptor parameter that is specified can be any of the descriptors that represent the controlling terminal (such as standard input [stdin], standard output [stdout], and standard error [stderr]). The service affects future access from any file descriptor in use for the terminal.

**Note:** You must consider redirection when choosing the file descriptor to specify.

5. The following table defines the processing of the SIGTTOU signal when the tcsetpgrp service is called from a background process against a controlling terminal:

<b>SIGTTOU processing</b>	<b>Expected behavior</b>
<b>Default or signal handler</b>	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
<b>Ignored or blocked</b>	The SIGTTOU signal is not sent. The function continues normally.

## Related services

- [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ” on page 777](#)
- [“setsid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ” on page 793](#)
- [“tcgetpgrp \(BPX1TGP, BPX4TGP\) — Get the foreground process group ID ” on page 931](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1TSP \(tcsetpgrp\) example” on page 1305](#).

# tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables

---

## Function

The tcsettables callable service sets the terminal session code page names and conversion tables to the specified values.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TST):</b>	31-bit
<b>AMODE (BPX4TST):</b>	64-bit
<b>ASC mode:</b>	Primary mode



<b>Operation</b>	<b>Environment</b>
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1TST,(File_descriptor,
              Termcp_length,
              Termcp_structure,
              Srctable,
              Trgtable,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TST with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor of the terminal for which the code page names and data conversion tables are to be set.

### Termcp\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of the fullword that contains the length of the Termcp\_structure. The Termcp\_structure is mapped by BPXYTCCP, and has a length of TCCP#LENGTH. See [“BPXYTCCP — Map the terminal control code page structure”](#) on page 1163.

### Termcp\_structure

Supplied parameter

#### Type:

Structure

#### Length:

Specified by Termcp\_length.

The name of an area that contains the code page information that is to be set. This structure contains the source (ASCII) code page name, target (EBCDIC) code page name, and control flags. The Termcp\_structure is mapped by the BPXYTCCP macro (see [“BPXYTCCP — Map the terminal control code page structure”](#) on page 1163).

### Srctable

Supplied parameter

#### Type:

Character string

**Character set:**

No restriction

**Length:**

256 bytes

The name of a field that contains a 256-byte data conversion table for the source-to-target (ASCII to EBCDIC) data conversion. The byte offset into this table corresponds to the character code from the source (ASCII) code page. The data value at each offset is the converted target (EBCDIC) character code.

**Trgtable**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

256 bytes

The name of a field that contains a 256-byte data conversion table for the target-to-source (EBCDIC to ASCII) data conversion. The byte offset into this table corresponds to the character code from the target (EBCDIC) code page. The data value at each offset is the converted source (ASCII) character code.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsettables service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsettables service stores the return code. The tcsettables service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The tcsettables service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	File_descriptor is an incorrect open file descriptor.
<b>EINTR</b>	A signal interrupted the call.

Return_code	Explanation
<b>EINVAL</b>	<p>One of the following error conditions exists:</p> <ul style="list-style-type: none"> <li>• The value of <i>Termcp_length</i> was not valid.</li> <li>• An incorrect combination of multi-byte code page names was specified in the <i>Termcp_structure</i>.</li> </ul> <p>One of the following applies:</p> <ul style="list-style-type: none"> <li>– The source code page that was specified in <i>TCCPSRCNAME</i> specified a supported ASCII multi-byte code page, and the <i>TCCPTRGNAME</i> did not specify a supported EBCDIC multi-byte code page.</li> <li>– The target code page that was specified in <i>TCCPTRGNAME</i> specified a supported EBCDIC multi-byte code page, and the <i>TCCPSRCNAME</i> did not specify a supported ASCII multi-byte code page.</li> </ul>
<b>EIO</b>	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
<b>ENODEV</b>	The terminal device driver does not support the forward code page names and tables CPCN capability.
<b>ENOTTY</b>	The file that is associated with the file descriptor is not a terminal device.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the tcsettables service stores the reason code. The tcsettables service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

**Attention:** Use this service carefully. By changing the code pages for the data conversion, you may cause unpredictable behavior in the terminal session if the actual data that is used for the session is not encoded to the specified source (ASCII) and target (EBCDIC) code pages.

1. Use the tcsettables callable service to send new code page names and data conversion tables to the terminal session data conversion point to change the data conversion environment.

The tcsettables callable service is used with terminal devices that support the forward code page names and tables CPCN capability. Use the tcgetcp (BPX1TGC, BPX4TGC) callable service to determine the terminal device CPCN capability.

2. The BPXYTCCP macro should be used to map the *Termcp\_structure* and define the equates for the flag byte values. Note the following about BPXYTCCP:
  - BPXYTCCP can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
  - The code page names that are contained in *TCCPSRCNAME* and *TCCPTRGNAME* must be terminated by a NUL (X'00') character.
  - The code page names that are contained in *TCCPSRCNAME* and *TCCPTRGNAME* are case sensitive.

3. The OCS remote-tty (rty) device driver supports this function. For OCS terminal sessions, the data conversion is performed by OCS outboard on the AIX® server system. Use the tcsettables service to specify new code pages and conversion tables that are to be used in the data conversion.

During its processing of the tcsettables service, the OCS rty device driver applies the new code page names when the outbound data queue is drained. When this occurs, the rty input data queue is also flushed, and the new conversion environment takes effect.

The *Srctable* and *Trgtable* parameters are used as follows:

- If the code page names that are specified in the Termcp\_structure are for supported double-byte data conversion the *SrcTable* and *TrgTable* arguments are not used. The following double-byte translation is supported for OCS sessions:

Source (ASCII) code page	Target (EBCDIC) code page
IBM-eucJP	IBM-939
IBM-932	IBM-939

- If TCCPSRCNAME specifies ISO8859-1 and TCCPTRGNAME specifies IBM-1047, OCS uses its own data conversion tables and the *Srctable* and *Trgtable* parameters are not used.
  - Otherwise the conversion tables in *Srctable* and *Trgtable* are used.
4. The data conversion for a terminal session is performed on a session (terminal file) basis. If you change the data conversion characteristics for one file descriptor, the new data conversion applies to all open file descriptors that are associated with this terminal file.
  5. Use the tcsettables callable service to notify the data conversion point to stop data conversion. This is done by setting the TCCPBINARY flag. If this flag is set, the source and target code page names (TCCPSRCNAME and TCCPTRGNAME, respectively) are not changed; the *Srctable* and *Trgtable* parameters are not used.
- Note:** Use this option carefully. When the data conversion is disabled, the z/OS shell cannot be used until the data conversion is reenabled, using valid code pages for the terminal session.
6. The TCCPFASTP flag is not used by the OCS rty device driver. The value of this flag has no effect and is ignored.
  7. The following table defines the processing of the SIGTTOU signal when the tcsettables service is called from a background process group against its controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.

## Characteristics and restrictions

None.

## Examples

See “BPX1TST (tcsettables) example” on page 1305.

## times (BPX1TIM, BPX4TIM) – Get process and child process times

### Function

The times callable service gathers information about processor time used by the current process or related processes.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TIM):</b>	31-bit
<b>AMODE (BPX4TIM):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1TIM, (Time_data,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TIM with the same parameters.

## Parameters

### Time\_data

Returned parameter

#### Type:

Structure

#### Length:

16 bytes

The name of a data area where the times service returns information about processor time used. This field is mapped by the BPXYTIMS macro. For the structure of the data area, see [“BPXYTIMS — Map the response structure for times”](#) on page 1168.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the service places its return value. The value that is returned is the number of clock ticks (hundredths of a second) that have elapsed since the current address space was last dubbed a process. If this value cannot be determined, the service returns -1.

### Return\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the times service stores the return code. The times service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The times service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
ERANGE	An overflow occurred while time values were being computed.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the times service stores the reason code. The times service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

Processor times for a child process that has ended are not added to the TIMSCUTIME and TIMSCSTIME of the parent process until the parent issues a wait or waitpid for that child process. See [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#) for more information about this subject.

**Related services**

- [“exec \(BPX1EXC, BPX4EXC\) — Run a program” on page 163](#)
- [“execmvs \(BPX1EXM, BPX4EXM\) — Run an MVS program ” on page 175](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“mvsprocclp \(BPX1MPC, BPX4MPC\) — Clean up kernel resources ” on page 484](#)
- [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end ” on page 984](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1TIM \(times\) example” on page 1303](#).

**MVS-related information**

The TIMSSTIME value that is returned by the times service is a portion of the total TCB time of the calling process—the portion that was spent processing z/OS UNIX services in the kernel address space. This TCB time is accumulated from the most recent time the MVS address space was dubbed a process (made eligible to issue z/OS UNIX callable services).

The TIMSUTIME value consists of the total processor time that has been accumulated by the calling address space in the current job-step. This includes all job step TCB and SRB time that was accumulated before the address space became a process, all SRB time that was accumulated after the address space became a process, and all TCB time that was accumulated after the address space became a process, except for the TCB time that was accumulated while the process was running in the kernel. The value of TIMSUTIME can be calculated as follows:

```
TIMSUTIME = <job-step SRB time> + <job-step TCB time> - TIMSSTIME
```

**Note:**

1. An MVS address space can be dubbed a process, undubbed (no longer a process), and then dubbed a process again in the same job step. The TIMSSTIME value for the address space in this case reflects only the kernel TCB time since the address space was last dubbed. The TIMSUTIME value, however, reflects TCB and SRB time for the entire life of the job step.
2. The exec service (BPX1EXC, BPX4EXC) and the execmvs service (BPX1EXM, BPX4EXM) cause new substeps in the current address space. Address-space-level processor time counters (in the address space control block) are reset. As long as the address space remains a process, the values from previous substeps are retained, and are included in values that are returned by the times service. However, if the mvspocclp service (BPX1MPC, BPX4MPC) is invoked to undub the process after the exec or execmvs service has been issued, subsequent invocations of the times service return processor times starting at the beginning of the new substep.
3. The times service reports an approximation of the usage by the system, and is not a completely accurate representation of the time used on behalf of the system and by the user. The function guarantees that the user time reported is ever increasing; it does not do the same for the system time.

## truncate (BPX1TRU, BPX4TRU) – Change the size of a file

---

**Function**

The truncate service changes the size of a file. The file is identified by a path name.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TRU):</b>	31-bit
<b>AMODE (BPX4TRU):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1TRU, (Pathname_length,
               Pathname,
               File_length,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4TRU with the same parameters.

## **Parameters**

### **Pathname\_length**

Supplied parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword that contains the length of the path name of the file whose size is to be changed.

### **Pathname**

Supplied parameter

#### **Type:**

Character string

#### **Character set:**

No restriction

#### **Length:**

Specified by the Pathname\_length parameter

The name of a field that contains the path name of the file. This field has the length that is specified in Pathname\_length.

Path names can begin with or without a slash:

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

### **File\_length**

Supplied parameter

#### **Type:**

Integer

#### **Length:**

Doubleword

The name of a doubleword that contains the number of bytes that are to be contained in the file after the size is changed.

This field is a doubleword to accommodate large files. For normal processing with a singleword value, the second word should be zero. The truncate service accepts only positive values.

### **Return\_value**

Returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword in which the truncate service returns 0 if the request is successful, or -1 if it is not successful.

### **Return\_code**

Returned parameter

#### **Type:**

Integer



**Length:**

Fullword

The name of a fullword in which the truncate service stores the return code. The truncate service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The truncate service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	The calling process does not have permission to search some component of the Pathname prefix; or write permission is denied on the file.
<b>EBUSY</b>	The file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
<b>EFBIG</b>	The File_length parameter is greater than the maximum file size limit for the process. The following reason code can accompany the return code: JRWriteBeyondLimit.
<b>EINVAL</b>	The file is not a regular file; or the File_length that is specified is either negative or greater than the maximum file size. The following reason codes can accompany the return code: JRTrNegOffset, JRTrNotRegFile.
<b>EISDIR</b>	The file is a directory.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters.
<b>ENOENT</b>	No file named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of the Pathname prefix is not a directory.
<b>EROFS</b>	The specified file is on a read-only file system. The following reason code can accompany the return code: JRTrMountedRO.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the truncate service stores the reason code. The truncate service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The truncate service changes the file size to File\_length bytes, beginning from the first byte of the file. If the file was originally larger than File\_length bytes, the data from File\_length to the original end of the file is removed. If the file was originally shorter than File\_length, bytes between the old and new lengths are read as zeros.
2. If File\_length is greater than the soft file size limit for the process, the request fails with EFBIG, and the SIGXFSZ signal is generated for the process.
3. Full blocks are returned to the file system, so that they can be used again.

4. A file may not be truncated if it is currently open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#) for details about the NFS share reservations.

## Related services

- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file” on page 249](#)

## Characteristics and restrictions

The file that is specified must be a regular file to which the calling process has write access.

## Examples

See [“BPX1TRU \(truncate\) example” on page 1304](#).

# ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal

---

## Function

The ttyname callable service obtains the pathname of the terminal that is associated with the file descriptor.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1TYN):</b>	31-bit
<b>AMODE (BPX4TYN):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1TYN, (File_descriptor,
               Terminal_name_length,
               Terminal_name)
```

AMODE 64 callers use BPX4TYN with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the file descriptor.

**Terminal\_name\_length**

Parameter supplied and returned

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains the size, in bytes, of the buffer that is referred to by Terminal\_name. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer that is specified should be the maximum length that the terminal\_name could be on output.

**Terminal\_name**

Returned parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Terminal\_name\_length parameter

The name of an area to which ttyname returns either the pathname of the terminal, terminated by a X'00', or a single byte of X'00' (null string), if the file descriptor is not valid or does not represent a terminal.

The length of Terminal\_name should be 1024 bytes long (PATH\_MAX+1), unless the pathname is known to be shorter.

**Usage notes**

1. This service does not return -1 to indicate a failure. If the file descriptor is incorrect, it returns a null string.
2. If Terminal\_name is an area smaller than the actual pathname of the terminal, the name is truncated.

**Related services**

- [“ttyname \(BPX2TYN, BPX4TYN\) \(X/Open version\) — Get the name of a terminal” on page 958](#)
- [“isatty \(BPX1ITY\) \(POSIX Version\) — Determine whether a file descriptor represents a terminal” on page 357](#)
- [“isatty \(BPX2ITY, BPX4ITY\) \(X/Open Version\) — Determine whether a file descriptor represents a terminal” on page 359](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1TYN \(ttyname\) example” on page 1306](#).

## ttyname (BPX2TYN, BPX4TYN) (X/Open version) – Get the name of a terminal

### Function

The ttyname callable service obtains the pathname of the terminal that is associated with the file descriptor.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX2TYN):</b>	31-bit
<b>AMODE (BPX4TYN):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX2TYN,(File_descriptor,
              Terminal_name_length,
              Terminal_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TYN with the same parameters.

### Parameters

#### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor.

#### Terminal\_name\_length

Parameter supplied and returned

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the size, in bytes, of the buffer that is referred to by Terminal\_name. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer that is specified should be the maximum length that the terminal\_name could be on output.

### Terminal\_name

Returned parameter

#### Type:

Character string

#### Character set:

No restriction

#### Length:

Specified by the Terminal\_name\_length parameter

The name of an area to which ttyname returns either the path name of the terminal, terminated by a X'00', or a single byte of X'00' (null string), if the file descriptor is not valid or does not represent a terminal.

The length of Terminal\_name should be 1024 bytes long (PATH\_MAX+1), unless the path name is known to be shorter.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the ttyname service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the ttyname service stores the return code. The ttyname service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The ttyname service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBADF</b>	The File_descriptor argument is not a valid open file descriptor.
<b>ENOTTY</b>	The File_descriptor argument is not associated with a terminal.

### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the ttyname service stores the reason code. The ttyname service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. This version of ttyname is XPG4 compliant.
2. If Terminal\_name is an area smaller than the actual pathname of the terminal, the name is truncated.

## Related services

[“isatty \(BPX1ITY\) \(POSIX Version\) — Determine whether a file descriptor represents a terminal” on page 357](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX2TYN \(ttyname\) example” on page 1306](#).

# umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask

## Function

The umask callable service changes the file mode creation mask of a process. The file mode creation mask is used by the security package to turn off permission bits in the mode parameter that is specified. Bit positions that are set in the file mode creation mask are cleared in the mode of the created file.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1UMK)</b>	31-bit
<b>AMODE (BPX4UMK)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1UMK, (File_mode_creation_mask,
               Return_value)
```

AMODE 64 callers use BPX4UMK with the same parameters.

## Parameters

**File\_mode\_creation\_mask**  
Supplied parameter

**Type:**

Structure

**Length:**

Fullword

The name of a fullword that contains the file mode creation mask. This mask turns off permission bits in a file's mode. File\_mode\_creation\_mask is mapped by the BPXYMODE macro (see [“BPXYMODE — Map the mode constants”](#) on page 1106).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the umask service returns the previous value of the file mode creation mask. This fullword has the same mapping as the File\_mode\_creation\_mask parameter.

**Usage notes**

1. The umask service changes the process's file creation mask. This mask controls file permission bits that are set whenever the process creates a file. File permission bits that are turned on in the file creation mask are turned off in the file permission bits of files that are created by the process. For example, if a call to the open (BPX1OPN, BPX4OPN) service specifies a "mode" argument with file permission bits, the process's file creation mask affects that argument: Bits that are on in the mask are turned off in the "mode" argument, and therefore in the mode of the created file.
2. Only the file permission bits of the new mask are used. For example, the type of file field in File\_Mode cannot be masked.

**Related services**

- [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory”](#) on page 420
- [“mkdirat \(BPX1MDA, BPX4MDA\) — Create a directory relative to a directory file descriptor”](#) on page 423
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1UMK \(umask\) example”](#) on page 1307.

## umount (BPX1UMT, BPX4UMT) — Remove a virtual file system

---

**Function**

The umount callable service unmounts or remounts a virtual file system. It removes a virtual file system from the file tree or it remounts a virtual file system to the file tree.

**Requirements****Operation****Environment****Authorization**

Supervisor or problem state, any PSW key

Operation	Environment
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1UMT)	31-bit
AMODE (BPX4UMT)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UMT,(File_system_name,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4UMT with the same parameters.

Parameters

**File\_system\_name**  
Supplied parameter

**Type**  
Character string

**Character set**  
Printable characters

**Length**  
44 bytes

The name of a 44-character field that contains the file system that is to be unmounted. The file system name must be left-aligned and padded with blanks.

**Flags**  
Supplied parameter

**Type**  
Structure

**Length**  
Fullword

The name of a fullword binary field that contains the unmount options.  
This field is mapped by the BPXYMTM macro. See [“BPXYMTM — Map the modes for mount and unmount”](#) on page 1110 for the contents of the macro.

**Return\_value**  
Returned parameter

**Type**  
Integer

**Length**  
Fullword



The name of a fullword in which the umount service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the umount service stores the return code. The umount service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The umount service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBUSY</b>	Honoring the request would require unmounting a file system that is still in use. The following reason codes can accompany the return code: JRFileSysWasReset, JRFsForceUmount, JRFsMustReset, JRFsParentFs, JRFsUnmountInProgress, JRIsFsRoot, and JRQuiesced.
<b>EINTR</b>	The call was interrupted by a signal. The following reason code can accompany the return code: JRSigDuringWait.
<b>EINVAL</b>	An incorrect parameter was specified. The file system name is not the name of a file system; an incorrect combination of flags was specified; a umount drain or remount request was specified in a sysplex; or an umount force was specified before an immediate umount was attempted. The following reason codes can accompany the return code: JRFileSysNotThere, JRInvalidParms, JRMustUmountImmed, JRQuiescing, JRNotSupInSysplex, JrRemntMode.
<b>EPERM</b>	The calling process is not a superuser. The following reason code can accompany the return code: JRUserNotPrivileged.

### Reason\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the umount service stores the reason code. The umount service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Usage notes

1. The caller must be APF-authorized or must be running for a user with mount authority.
2. A file system that has file systems mounted on it can be remounted but cannot be unmounted. Before the file system can be unmounted, all children file systems must be unmounted first.
3. A reset request can stop only an umount service drain request. It has no effect if it is issued when there is no umount request outstanding. Currently, umount service drain requests are not supported in a sysplex environment. If such a request is issued in a sysplex, the following behavior is exhibited:
  - If there is no activity in the file system, the drain request performs the unmount, but it behaves like a umount normal. (Where a normal request specifies that if no user is accessing any of the files in the

specified file system, the system processes the umount request. Otherwise, the system rejects the umount request. This is the default.)

- If there is activity in the file system, the drain request returns a return value of -1, with return code EINVAL and reason code JRNotSupInSysplex.
4. A umount service request with no other options specified succeeds only if the unmount can be processed immediately. Otherwise, an EBUSY is returned.
  5. MTMREMOUNT is specified to change the mount mode between read-only and read/write. If neither MTMRO nor MTMRDWR is specified, the mode is set to the opposite of its current state. If a mode is specified, it must be the opposite of the current state.
  6. MTMSAMEMODE is specified to remount the file system without changing the mount mode. This function can be used to attempt to regain use of a file system that has had I/O errors. If MTMREAD or MTMRDWR is also specified, the specified mode must be the current mode.
  7. Before a file system is remounted (using any method other than MTMSAMEMODE), any open FIFO files must be closed, or the remount attempt is rejected with EINVAL, JrFIFIInFileSys.
  8. If the file system that is to be unmounted is the root file system, the IMMED option must be specified.
  9. The MTMDETACH flag specifies that the file system and any file systems that are mounted under it are not available for new accesses. Any active references to those file systems are allowed to continue until there are no more references. At that point, the file system is unmounted. The **umount** command immediately returns and processing continues asynchronously.

## Related services

[“mount \(BPX1MNT\) — Make a file system available ” on page 444](#)

## Characteristics and restrictions

In order to unmount a file system, the requester must be an authorized program, or must be running for a user with appropriate privileges (see [“Authorization” on page 8](#)).

## Examples

See [“BPX1UMT \(umount\) example” on page 1307](#).

# uname (BPX1UNA, BPX4UNA) — Obtain the name of the current operating system

---

## Function

The uname callable service obtains information about the z/OS UNIX system that the caller is running on.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1UNA)</b>	31-bit
<b>AMODE (BPX4UNA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts

<b>Operation</b>	<b>Environment</b>
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1UNA, (Data_area_length,
               Data_area_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4UNA with the same parameters. The Data\_area\_address parameter is a doubleword.

## Parameters

### Data\_area\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the data area that is pointed to by Data\_area\_address. The area must be at least the length of UTSN#LENGTH. For a mapping of this data area, refer to [“BPXYUTSN — Map the response structure for uname”](#) on page 1172.

### Data\_area\_address

Returned parameter

#### Type

Address

#### Length

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the area in which the system information is to be returned. For a mapping of this data area, refer to [“BPXYUTSN — Map the response structure for uname”](#) on page 1172.

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the uname service returns a nonnegative value if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type

Integer

#### Length

Fullword

## unlink (BPX1UNL, BPX4UNL)

The name of a fullword in which the uname service stores the return code. The uname service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The uname service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
EFAULT	The pointer to the UTSN from the invoker is bad. The following reason code can accompany the return code: JRBadAddress.
EINVAL	The passed length of the invoker UTSN is not valid. The following reason code can accompany the return code: JROK.

### Reason\_code

Returned parameter

### Type

Integer

### Length

Fullword

The name of a fullword in which the uname service stores the reason code. The uname service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Characteristics and restrictions

None.

## Examples

See “[BPX1UNA \(uname\) example](#)” on page 1307.

# unlink (BPX1UNL, BPX4UNL) — Remove a directory entry

---

## Function

The unlink service removes a directory entry. A directory entry can be identified by a path name to a file, a link name to a file, or a symbolic link.

If a link to a file is removed, and the link count becomes zero, and no other process has the file open, the file itself is deleted.

## Requirements

Operation	Environment
Authorization	Supervisor or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1UNL)	31-bit
AMODE (BPX4UNL)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked

## Operation

## Environment

### Control parameters

All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1UNL, (Name_length,
               Name,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4UNL with the same parameters.

## Parameters

### Name\_length

Supplied parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of Name.

### Name

Supplied parameter

#### Type

Character string

#### Character set

No restriction

#### Length

Specified by the Name\_length parameter

The name of a field, of length Name\_length, that contains the name of the directory entry that is to be removed. Name can be a path name to a file, a link name to a file, or a symbolic link name. The path name was specified when the file was created (see [“open \(BPX1OPN, BPX4OPN\) – Open a file”](#) on page 511). The link name was specified when a link to the file was created (see [“link \(BPX1LNK, BPX4LNK\) – Create a link to a file”](#) on page 383), or when the symbolic link was created (see [“symlink \(BPX1SYM, BPX4SYM\) – Create a symbolic link to a path name”](#) on page 900).

### Return\_value

Returned parameter

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the unlink service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type

Integer

**Length**

Fullword

The name of a fullword in which the unlink service stores the return code. The unlink service returns `Return_code` only if `Return_value` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The unlink service can return one of the following values in the `Return_code` parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>• The calling process does not have permission to search some component of the path name or does not have write permission for the directory that contains the link that is to be removed.</li> <li>• The <code>S_ISVTX</code> flag is set for the parent directory of the file that is to be removed and the caller is not the owner of the file or of the parent directory. The caller also does not have appropriate privileges. (See <a href="#">“Authorization”</a> on page 8.)</li> </ul>
<b>EBUSY</b>	The file cannot be unlinked because it is being used by the system or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
<b>EINVAL</b>	The Name parameter is incorrect. It contains a null character.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Name.
<b>ENAMETOOLONG</b>	Name is longer than 1023 characters, or some component of the path name is longer than 255 characters. Name truncation is not supported.
<b>ENOENT</b>	Name was not found, or no name was specified. The following reason code can accompany the return code: <code>JRUnlNoEnt</code> .
<b>ENOTDIR</b>	Some component of the path name prefix is not a directory.
<b>EPERM</b>	Name refers to a directory. Directories cannot be removed with unlink. The following reason code can accompany the return code: <code>JRUnlDir</code> .
<b>EROFS</b>	The link that is to be removed is on a read-only file system. The following reason code can accompany the return code: <code>JRUnlMountRO</code> .

**Reason\_code**

Returned parameter

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the unlink service stores the reason code. The unlink service returns `Reason_code` only if `Return_value` is -1. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. If the name that is specified refers to a symbolic link, the symbolic link file that is named by Name is deleted.
2. If the unlink service request is successful and the link count becomes zero, the file is deleted. The contents of the file are discarded, and the space it occupied is freed for reuse. However, if another

process (or more than one) has the file open or in use when the last link is removed, the file is not removed until the last process closes it.

3. When the unlink service is successful in removing the directory entry and decrementing the link count, whether or not the link count becomes zero, it returns control to the caller with Return\_value set to 0. It updates the change and modification times for the parent directory and the change time for the file itself (unless the file is deleted).
4. Directories cannot be removed with the unlink service. To remove a directory, refer to [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory” on page 703](#).
5. If the S\_ISVTX flag is set for the parent directory of the file that is to be unlinked, one of the following conditions must be true, or the request will fail with EACCES:
  - The caller is the owner of the file to be unlinked.
  - The caller is the owner of the parent directory.
  - The caller has appropriate privileges (see [“Authorization” on page 8](#)).
6. A file cannot be unlinked if it is open by a remote NFS client with a share reservation that would prevent the file from being opened for writing. Refer to [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#) for details about the NFS Share reservations.

## Related services

- [“close \(BPX1CLO, BPX4CLO\) — Close a file” on page 121](#)
- [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file” on page 383](#)
- [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor” on page 387](#)
- [“open \(BPX1OPN, BPX4OPN\) — Open a file” on page 511](#)
- [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory” on page 690](#)
- [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory” on page 703](#)
- [“unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor” on page 969](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1UNL \(unlink\) example” on page 1308](#).

# unlinkat (BPX1ULA, BPX4ULA) — Remove a directory or directory entry relative to a directory file descriptor

---

## Function

The unlinkat service removes a directory or a directory entry relative to a directory file descriptor. The target can be identified by a path name to a file or directory, a link name to a file or directory, or a symbolic link. If a link to a file is removed, and the link count becomes zero, and no other process has the file open, the file itself is deleted. A directory must be empty for unlinkat to succeed.

## Requirements

### Operation

### Environment

### Authorization

Supervisor or problem state, any PSW key

<b>Operation</b>	<b>Environment</b>
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1ULA)</b>	31-bit
<b>AMODE (BPX4ULA)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1ULA, (Name_length,
               Name,
               Dirfd,
               Flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4ULA with the same parameters.

## Parameters

### Name\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of Name.

### Name

Supplied parameter

#### Type

Character string.

#### Character set

No restriction.

#### Length

Specified by the Name\_length parameter.

The name of a field that contains the pathname of the directory or directory entry to be removed. The length of this field is specified in Name\_length. If AT\_REMOVEDIR is specified in the Flags parameter the pathname must represent a directory. Otherwise the pathname must represent a directory entry.

### Dirfd

Supplied parameter

#### Type

Integer.

#### Length

Fullword.



The name of a fullword that contains a file descriptor for the directory to be used for the unlinkat request. The special value AT\_FDCWD specified with a relative pathname, indicates to use the current working directory, otherwise a relative pathname will be interpreted relative to the directory referred to by this file descriptor. If Name represents an absolute path, this parameter is ignored.

### Flags

Supplied parameter

#### Type

Integer.

#### Length

Fullword.

The name of a fullword field to further define the unlinkat request. Flags can either be 0 or include the following, which are defined in the BPXYFCTL macro.

#### AT\_REMOVEDIR

When specified this indicates that the target of the operation is a directory. If not specified then the target of the operation is a directory entry.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unlinkat service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unlinkat service stores the return code. The unlinkat service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The unlinkat service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	One of the following conditions occurred: <ul style="list-style-type: none"> <li>The calling process does not have permission to search some component of the path name or does not have write permission for the directory that contains the link that is to be removed.</li> <li>The S_ISVTX flag is set for the parent directory of the file or directory that is to be removed, and the caller is not the owner of the file or of the parent directory; nor does the caller have appropriate privileges (see <a href="#">“Authorization” on page 8</a>).</li> </ul>
<b>EBADF</b>	The file descriptor that was specified for <i>Dirfd</i> is incorrect.

Return_code	Explanation
<b>EBUSY</b>	<p>The resource is busy. The following reason codes can accompany the return code:</p> <ul style="list-style-type: none"> <li>• <b>JrRootNode</b>. The requested operation cannot be done on a root directory.</li> <li>• <b>JrShrConflict</b>. The file is open by a remote NFS client with a share reservation that conflicts with the requested operation.</li> <li>• <b>JrFileIsBound</b>. The file or directory cannot be removed because it is in use by another process.</li> </ul>
<b>EINVAL</b>	<p>One of the input parameters was not valid. The following reason codes can accompany the return code:</p> <ul style="list-style-type: none"> <li>• <b>JrInvalidParms</b>. A null character was encountered in the path name or an invalid flag was specified in the <i>Flags</i> parameter.</li> <li>• <b>JrDotOrDotDot</b>. Dot (.) or dot-dot (..) was specified in the path name.</li> </ul>
<b>EISDIR</b>	<p>The <i>Name</i> specified is a directory and <b>AT_REMOVEDIR</b> was not specified in the <i>Flags</i> parameter. The following reason code can accompany the return code: <b>JrUnlDir</b>.</p>
<b>ELOOP</b>	<p>A loop exists in symbolic links that were encountered during resolution of the <i>Name</i> argument. This error is issued if more than 24 symbolic links are detected in the resolution of <i>Name</i>.</p>
<b>ENAMETOOLONG</b>	<p>Name is longer than 1023 characters, or some component of the path name is longer than 255 characters. File name truncation is not supported.</p>
<b>ENOENT</b>	<p>Name was not found, or no name was specified. The following reason code can accompany the return code:</p> <ul style="list-style-type: none"> <li>• <b>JrUnlNoEnt</b>. The file to be unlinked does not exist</li> <li>• <b>JrFileNotThere</b>. The requested directory does not exist.</li> </ul>
<b>ENOTDIR</b>	<p>Parameter is not a directory. The following reason codes can accompany the return code:</p> <ul style="list-style-type: none"> <li>• <b>JrPathNotDir</b>. The <i>Name</i> specified is not a directory.</li> <li>• <b>JrAtNotDirFd</b>. The <i>Dirfd</i> parameter does not refer to a directory.</li> </ul>
<b>EROFS</b>	<p>The file or directory that is to be removed is on a read-only file system. The following reason code can accompany the return code: <b>JrUnlMountRO</b>.</p>
<b>Reason_code</b>	
Returned parameter	
<b>Type:</b>	
Integer	
<b>Length:</b>	
Fullword	
<p>The name of a fullword in which the unlinkat service stores the reason code. The unlinkat service returns <i>Reason_code</i> only if <i>Return_value</i> is -1. <i>Reason_code</i> further qualifies the <i>Return_code</i> value. For a list of reason codes, see <a href="#">Reason codes</a> in <i>z/OS UNIX System Services Messages and Codes</i>.</p>	

## Usage notes

1. If the name that is specified refers to a symbolic link, the symbolic link file that is named by *Name* is deleted.

2. If the unlinkat service request is successful and the link count becomes zero, the file is deleted. The contents of the file are discarded, and the space it occupied is freed for reuse. However, if another process (or more than one) has the file open or in use when the last link is removed, the file is not removed until the last process closes it.
3. When the unlinkat service is successful in removing the directory entry and decrementing the link count, whether or not the link count becomes zero, it returns control to the caller with Return\_value set to 0. It updates the change and modification times for the parent directory and the change time for the file itself (unless the file is deleted).
4. Directories can only be removed by using the unlinkat service if AT\_REMOVEDIR was specified in the Flags parameter. The target directory must be empty.
5. If the S\_ISVTX flag is set for the parent directory of the file or directory that is to be unlinked, one of the following conditions must be true, or the request will fail with EACCES:
  - The caller is the owner of the file or directory to be unlinked.
  - The caller is the owner of the parent directory.
  - The caller has appropriate privileges (see “Authorization” on page 8).
6. A file cannot be unlinked if it is currently open by a remote NFS client with a share reservation that would prevent the file from being opened for writing. Refer to “open (BPX1OPN, BPX4OPN) — Open a file” on page 511 for details about the NFS Share reservations.

## Related services

- “link (BPX1LNK, BPX4LNK) — Create a link to a file ” on page 383
- “linkat (BPX1LNA, BPX4LNA) — Make a link using at directory relative to a directory file descriptor ” on page 387
- “mkdir (BPX1MKD, BPX4MKD) — Make a directory ” on page 420
- “mkdirat (BPX1MDA, BPX4MDA) — Create a directory relative to a directory file descriptor ” on page 423
- “open (BPX1OPN, BPX4OPN) — Open a file” on page 511
- “rename (BPX1REN, BPX4REN) — Rename a file or directory ” on page 690
- “rmdir (BPX1RMD, BPX4RMD) — Remove a directory ” on page 703
- “unlink (BPX1UNL, BPX4UNL) — Remove a directory entry ” on page 966

## Characteristics and restrictions

None.

## Examples

See “BPX1ULA (unlinkat) example” on page 1306 and “BPX4ULA (unlinkat) example” on page 1399.

# unlockpt (BPX1UPT, BPX4UPT) — Unlock a pseudoterminal manager and subsidiary pair

---

## Function

The unlockpt callable service unlocks the subsidiary pseudoterminal device that is associated with the manager to which the file descriptor refers.

**Note:** Because access to pseudoterminals is granted by changing ownership during the first subsidiary open, and that caller must have the same UID as the manager opener, neither the grantpt nor the unlockpt services are functionally required. They are provided in order to be compatible with XPG4, and for ported programs that may use them.

## Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1UPT)</b>	31-bit
<b>AMODE (BPX4UPT)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1UPT,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UPT with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor for the terminal.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unlockpt service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unlockpt service stores the return code. The unlockpt service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes](#)

(errno) in *z/OS UNIX System Services Messages and Codes* The unlockpt service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Either a grantpt has not yet been issued, or an unlockpt has already been issued. An unlockpt must be issued after a grantpt, and can only be issued once.
<b>EBADF</b>	The File_descriptor parameter does not specify a file descriptor that is open for writing.
<b>EINVAL</b>	The file descriptor is not associated with a manager pseudoterminal device.

#### Reason\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unlockpt service stores the reason code. The unlockpt service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

### Characteristics and restrictions

None.

### Examples

See [“BPX1UPT \(unlockpt\) example”](#) on page 1308.

## unquiesce (BPX1UQS, BPX4UQS) – Unquiesce a file system

### Function

The unquiesce callable service unquiesces a file system, making the files in it available for use again. The backup of the data in the file system is complete.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1UQS):</b>	31-bit
<b>AMODE (BPX4UQS):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Serialization:</b>	Enabled for interrupts
<b>Locks:</b>	No locks held
<b>Control parameters:</b>	All parameters addressable in Primary

## Format

The syntax format is as follows:

```
CALL BPX1UQS,(File_system_name,  
              Unquiesce_parms,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4UQS with the same parameters.

## Parameters

### File\_system\_name

Supplied parameter

#### Type:

Character string

#### Character set:

Printable characters

#### Length:

44 bytes

The name of a 44-character field that contains the file system name. The name should be left-justified in the field and padded with blanks.

### Unquiesce\_Parms

Supplied parameter

#### Type:

Structure

#### Length:

Fullword

The name of a fullword binary field that contains the unquiesce service options. This field is mapped by the BPXYMTM macro. Refer to [“BPXYMTM — Map the modes for mount and unmount”](#) on page 1110 for the unquiesce service options that are available.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unquiesce service returns 0 if the request is successful, or -1 if it is not successful.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the unquiesce service stores the return code. The unquiesce service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errnos\)](#) in *z/OS UNIX System Services Messages and Codes*. The unquiesce service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EBUSY</b>	The file system that was specified was not quiesced by this user, and force was not specified in Unquiesce_parms. The following reason code can accompany the return code: JRInvalidRequester.
<b>EINVAL</b>	An incorrect parameter was specified. Verify that only the force bit in Unquiesce_parms was specified, that File_system_name is correct, and that File_system_name is for a quiesced file system. The following reason codes can accompany the return code: JRFileSysNotThere, JRInvalidParms, and JRNotQuiesced.
<b>EPERM</b>	The user cannot request this service, because it lacks the permission required to do so. The following reason code can accompany the return code: JRUserNotPrivileged.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the unquiesce service stores the reason code. The unquiesce service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

The unquiesce service makes a file system available for use again following a previous quiesce request.

**Related services**

[“quiesce \(BPX1QSE, BPX4QSE\) — Quiesce a file system” on page 650](#)

**Characteristics and restrictions**

1. In order to unquiesce a file system, the requester must be a superuser. This is the same authority that is required to mount or quiesce a file system.
2. In a sysplex, an unquiesce will result in the file system being mounted on any system that did not have the file system mounted at that time. This situation could occur if a system joined the sysplex during the period of time that the file system was in a quiesced state.

**Examples**

See [“BPX1UQS \(unquiesce\) example” on page 1308](#).

## unshare (BPX1UNS, BPX4UNS) — Run program in a new namespace

---

**Function**

The unshare service disassociates a process from each namespace whose type is specified by the Flags parameter and moves it into a newly created namespace. For namespaces other than PID namespaces, the current process is moved into the new namespace as a result of the unshare service. For PID namespaces, the first child that is created after the unshare request by the current process is created in

a new PID namespace. Subsequent children of the same calling process are also associated with this new PID namespace.

For the macro, see [“BPXYCLNP— Map clone syscall parameters” on page 1062](#).

Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1UNS)	31-bit
AMODE (BPX4UNS)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UNS,(Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UNS with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Flags

Supplied parameter.

Type

Integer

Length

Fullword

The name of a fullword field that specifies which namespaces are to be disassociated by the process and a new namespace created. The flags are mapped by the CLONE\_FLAGS structure that is contained within the BPXYCLNP macro. For more information, see [“BPXYCLNP— Map clone syscall parameters” on page 1062](#).

Zero or more of the following supported flags can be specified on the unshare service.

Flag	Description
0	No-op. No changes are made.
CLONE_NEWIPC	Unshare the process from the IPC namespace and move into a new IPC namespace.
CLONE_NEWNS	Unshare the process from the mount namespace and move into a new mount namespace.



Flag	Description
CLONE_NEWPID	Unshare the PID namespace so the subsequent child of the current process will create a new PID namespace.
CLONE_NEWUTS	Unshare the process from the UTS namespace and move into a new UTS namespace.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the unshare service returns 0 if the request is successful or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the unshare service stores the return code. The unshare service returns `Return_code` only if `Process_ID` is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. A list of possible return codes follows.

**Return\_code****Explanation****EINVAL**

An input parameter was not valid.

The following reason code can accompany the return code:  
JRUnsupportedFlag, JrMultiProc, JrMultiPidUnshares.

**ENOSPC**

A system limit was reached.

- The limit on the number of namespaces would be exceeded.
- Creating a new PID namespace as requested by CLONE\_NEWPID would cause the nesting depth limit of PID namespaces to be exceeded.

The following reason code can accompany the return code:  
JRMaxNamespace, JrMaxNamespaceNestin.

**EPERM**

The calling process does not have appropriate privileges.

- The user is not a superuser and is not permitted to the CONTAINERS resource in the UNIXPRIV class.

**EAGAIN**

The resources required to let another process be created are not available or you have reached the maximum number of processes that you can run.

The following reason code can accompany the return code:  
JrNoSecurityProduct.

**EMVSSAF2ERR**

An error occurred in the security product.

The following reason code can accompany the return code:  
JrSAFInternal.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the unshare service returns 0 if the request is successful, or -1 if it is not successful.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the unshare service stores the reason code. The **unshare** service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes for unshare**

1. When the input flags specified are zero, the unshare is treated as a no-op, and will indicate success on return.
2. Flags that are supported by unshare are CLONE\_NEWNS, CLONE\_NEWPID, CLONE\_NEWIPC, and CLONE\_NEWUTS. Any other flag will result in the operation failing with an EINVAL error.
3. When the unshare requests the process to be moved to one or more new namespaces (nonzero unshare flags specified), the caller must be authorized by being a superuser or have at least READ access to the CONTAINERS resource in the UNIXPRIV class.
4. For unshare syscalls that specify a flag (CLONE\_NEWNS, CLONE\_NEWIPC, and CLONE\_NEWUTS), the requested namespace is created and the current process is moved into the new namespace after the syscall completes. For unshare syscalls that specify the CLONE\_NEWPID flag, the new PID namespace is not immediately created and the current process never changes namespaces. Instead, a new PID namespace is created when the first subsequent child of the calling process is created. Any additional child processes of the calling process is created in the namespace that was created by the first child after the unshare syscall.

**Related services**

- [“clone \(BPX1CLN, BPX4CLN\) — Create a child process” on page 114](#)
- [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#)
- [“setns \(BPX1SNS, BPX4SNS\) — Reassociate thread with a namespace” on page 771](#)
- [“spawn \(BPX1SPN, BPX4SPN\) — Spawn a process ” on page 869](#)

**Examples**

For examples that use the unshare callable service, see [“BPX1UNS \(unshare\) example” on page 1308](#) and [“BPX4UNS \(unshare\) example” on page 1401](#).

**Characteristics and restrictions**

Following is a list of characteristics or restrictions for the unshare service:

- There is a limit on the maximum number of namespaces (all types combined). It is set to one half of the maximum process limit. It is a static limit that is not affected by any changes that are made to MAXPROCSYS.
- PID namespaces can be nested, thus forming a hierarchical tree. There is a limit to the nesting depth of PID namespaces set to four namespaces levels underneath the root namespace.
- A process can only perform the unshare operation once. Subsequent unshare Clone\_Newpid or clone Clone\_Newpid requests will result in a failure.

## utime (BPX1UTI, BPX4UTI) – Set file access and modification times

---

### Function

The utime callable service sets the access and modification times of a file.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1UTI):</b>	31-bit
<b>AMODE (BPX4UTI):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1UTI,(Pathname_length,
               Pathname,
               Newtimes,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4UTI with the same parameters. The Newtimes parameter is two doublewords.

### Parameters

#### Pathname\_length

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the length of the fully qualified name (path name) of the file. The path name can be up to 1023 bytes long.

**Pathname**

Supplied parameter

**Type:**

Character string

**Character set:**

No restriction

**Length:**

Specified by the Pathname\_length parameter

The name of a field of length Pathname\_length that contains the path name of the file.

Path names can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

**Newtimes**

Supplied parameter

**Type:**

Structure

**Length:**

Doubleword (two doublewords)

The name of a doubleword or two doublewords that contain the access and modification times for the file. The first fullword (doubleword) contains the new access time, and the second fullword (doubleword) contains the new modification time. These times can be retrieved with [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name”](#) on page 893 or [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor”](#) on page 238.

- Times are specified as the number of seconds that have elapsed between 00:00 a.m. on January 1, 1970, and the desired time. The times must be specified as nonnegative values other than -1 (see this topic for the special case of -1). AMODE 64 callers must specify each time as a doubleword. Times beyond the year 2038 require more than a fullword.
- In order to request that the current time be used for both access and modification times, specify X'FFFFFFFF' or X'FFFFFFFFFFFFFFFF' (-1) in either or both words (doublewords) of this field. The current time in the file's status is also updated.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the utime service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the utime service stores the return code. The utime service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in

*z/OS UNIX System Services Messages and Codes* The utime service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EACCES</b>	One of the following is true: <ul style="list-style-type: none"> <li>• The process does not have search permission for some component of the Pathname prefix.</li> <li>• Newtimes equals the current time; the effective ID does not match the file's owner; the process does not have write permission for the file; and the process does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).</li> </ul>
<b>EINVAL</b>	The argument that was supplied is incorrect. The following reason code can accompany the return code: JRNegativeValueInvalid.
<b>ELOOP</b>	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
<b>ENAMETOOLONG</b>	The length of the path name is greater than 1023, or some component of the fully qualified name is longer than 255 bytes. This could be as a result of encountering a symbolic link during resolution of Pathname, where the substituted string is longer than 1023 characters.
<b>ENOENT</b>	No file named Pathname was found; or Pathname was blank. The following reason code can accompany the return code: JRFileNotThere.
<b>ENOTDIR</b>	Some component of the path name prefix is not a directory.
<b>EPERM</b>	The Newtimes value did not specify the current time; the effective user ID of the calling process does not match the owner of the file; and the calling process does not have appropriate privileges (see <a href="#">“Authorization”</a> on page 8).
<b>EROFS</b>	Pathname is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFs.

#### **Reason\_code**

Returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword in which the utime service stores the reason code. The utime service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

#### **Usage notes**

- For an AMODE 31 version that allows 64-bit times, see [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory”](#) on page 82.

#### **Related services**

- [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor”](#) on page 238
- [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name”](#) on page 893

## Characteristics and restrictions

None.

## Examples

See “BPX1UTI (utime) example ” on page 1309.

# wait (BPX1WAT, BPX4WAT) – Wait for a child process to end

---

## Function

The wait callable service obtains the status of a child process that has ended or stopped. You can use the wait service to obtain the status of a process that is being debugged with the ptrace facilities. The term *child* refers to children that are created by the fork service, as well as processes that are attached by ptrace.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1WAT):</b>	31-bit
<b>AMODE (BPX4WAT):</b>	64-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1WAT,(Process_ID,
               Options,
               Status_field_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4WAT with the same parameters. The Status\_field\_address parameter is a doubleword.

## Parameters

### Process\_ID

Supplied parameter

### Type:

Integer

### Length:

Fullword

The name of a fullword that contains a value that indicates the event that the caller is waiting upon:

- A value greater than zero is assumed to be a process ID. The caller waits for the child or debugged process with that specific process ID to end or to stop.
- A value of zero specifies that the caller is waiting for any children or debugged processes with a process group ID equal to the caller's to end or to stop.
- A value of -1 specifies that the caller is waiting for any of its children or debugged processes to end or to stop.
- If the value is negative and less than -1, its absolute value is assumed to be a process group ID. The caller waits for any children or debugged processes with that process group ID to end or to stop.

### Options

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the wait options for this invocation of the wait service. The wait options that are specified affect the actions that are taken by the wait service, as described in this topic. These options can be specified separately or in combination. A zero value for the wait options implies that the wait service performs its default processing; that is, it waits for a child process to end. The following flags defined in the BPXYCONS macro are the allowable wait options (see [“BPXYCONS — Constants used by services”](#) on page 1062).

Constant	Description
<b>WNOHANG</b>	The wait service does not suspend execution of the calling process if status is not immediately available for one of the child processes that is specified by Process_ID.
<b>WUNTRACED</b>	The wait service also returns the status of any child processes that are specified by Process_ID that are stopped, and whose status has not yet been reported since they stopped. If this option is not specified, the wait service returns only the status of processes that end.
<b>WCONTINUED</b>	The wait service returns the status for any continued child process that is specified by Process_ID whose status has not yet been reported since it continued from a job control stop.

### Status\_field\_address

Returned parameter

#### Type:

Address

#### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a fullword in which this service is to place the status value for the child process that ended or stopped. The status value can be analyzed with the status value map BPXYWAST. For a description of this mapping, see [“BPXYWAST — Map the wait status word”](#) on page 1172. The status value is returned only if status is available for a child or debugged process, and the address specified in this field is not zero.

### Return\_value

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the wait service returns the process ID of the child that the status information applied to, if the request is successful, or -1 if it is not successful, or 0 if WNOHANG was specified and there is at least one process whose status information is not available.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the wait service stores the return code. The wait service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The wait service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>ECHILD</b>	The caller has no appropriate child process; that is, no child process whose status has not already been obtained through earlier calls to wait meets the criteria for waiting.
<b>EFAULT</b>	One of the parameters that was specified contained the address of a storage area that is not accessible to the caller. The following reason code unique to this service can accompany this return code: JRBadExitStatusAddr.
<b>EINTR</b>	The calling process received a signal before the completion of an event that would cause the wait service to return. The service was interrupted by a signal. In this case, the value that is contained in Status_field_address is undefined.
<b>EINVAL</b>	The value of the option is not valid.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the wait service stores the reason code. The wait service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. The wait service suspends execution of the calling thread until one of the requested child or debugged processes ends, or until it obtains information about the process that ended. If a child or debugged process has already ended, but its status has not been reported when wait is called, the routine immediately returns with that status information to the caller.

If the WUNTRACED option is specified, the foregoing also applies for stopped children and stopped debugged processes.

2. The wait service always returns status for stopped debugged processes, even if WUNTRACED is not specified.

If status is available for one or more processes, the order in which the status is reported is unspecified.



3. If the wait service is invoked simultaneously from multiple threads within the same process, the following behavior should be noted:

- When multiple threads issue a fork call followed by a call to the wait service to wait for any child process to end, the status that is received by each thread might not be the status of the child that was created by that thread. If a thread wants to receive the status of the child that it created, the thread should specify the returned child Process Id when it calls the wait service to wait for the child process to end.
- If the wait service is called from multiple threads requesting status for the same process, the thread that receives the status is not specified when the process ends or stops. The thread that does not receive the status is returned to with a return value of -1 and a return code of ECHILD.

**Note:** A debugged process is one that is being monitored for debugging purposes with the ptrace service.

## Related services

- “[\\_exit \(BPX1EXI, BPX4EXI\)](#) — End a process and bypass the cleanup ” on page 180
- “[fork \(BPX1FRK, BPX4FRK\)](#) — Create a new process ” on page 227
- “[pause \(BPX1PAS, BPX4PAS\)](#) — Suspend a process pending a signal” on page 539
- “[ptrace \(BPX1PTR, BPX4PTR\)](#) — Control another process for debugging” on page 620

## Characteristics and restrictions

None.

## Examples

See “[BPX1WAT \(wait\) example](#) ” on page 1309.

# wait-extension (BPX1WTE, BPX4WTE) — Obtain status information for children

---

## Function

The wait-extension callable service allows the calling process to obtain status information for its child processes.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1WTE)</b>	31-bit
<b>AMODE (BPX4WTE)</b>	64-bit
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The format is as follows:

```
CALL BPX1WTE(Function_code,
             Idtype,
             Id,
             Stat_loc_ptr,
             Options,
             Info_area_ptr,
             Return_value,
             Return_code,
             Reason_code)
```

AMODE 64 callers use BPX4WTE with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

## Parameters

### Function\_code

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains a value that indicates the function to perform. If the value is `#wait3`, the `wait3()` function is performed. If the value is `#waitid`, the `waitid()` function is performed. If the value is `#wait4`, the `wait4()` function is performed. The constants `#wait3`, `#waitid`, and `#wait4` are defined in macro `BPXYCONS`.

### Idtype

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains a value that indicates what type of children to wait for. This parameter is valid only when `Function_code` is `#waitid`. It can be one of the following values:

#### P\_PID

`waitid()` waits for the child with a process ID that is equal to the value that is specified in the **id** parameter.

#### P\_PGID

`waitid()` waits for the child with a process group ID that is equal to the value that is specified in the **id** parameter.

#### P\_ALL

`waitid()` waits for any children. The **id** parameter is ignored.

The `P_` constants are defined in the `BPXYCONS` macro.

### Id

Supplied parameter.

#### Type

Integer.

#### Length

Fullword.

The name of a fullword that contains a value that defines which children are to be waited for. This value is valid only when the `function_code` is `#waitid` or `#wait4`. Together with `Idtype`, `Id` is used to determine which children are to be waited for.

If the function code is `#wait4`, then the value is interpreted as follows:

- A value greater than zero is assumed to be a process ID. The caller waits for the child or debugged process with that specific process ID to end or to stop.
- A value of zero specifies that the caller is waiting for any children or debugged processes with a process group ID equal to the caller's to end or to stop.
- A value of -1 specifies that the caller is waiting for any of its children or debugged processes to end or to stop.
- If the value is negative and less than -1, its absolute value is assumed to be a process group ID. The caller waits for any children or debugged processes with that process group ID to end or stop.

If the function code is `#waitid`, then the value might be a process ID or process group ID and `Idtype` further defines which children are to be waited for.

### **Stat\_loc\_ptr**

Supplied parameter.

#### **Type**

Pointer.

#### **Length**

Fullword (doubleword).

The name of a fullword (doubleword) that contains the address of a fullword in which this service is to place the status value for the child process whose status is available.

This parameter is valid only when the `function_code` is `#wait3` or `#wait4`.

If the wait-extension service returns because the status of a child process is available, and if `Stat_loc_ptr` is not a null pointer, information is stored in the location that is pointed to by `Stat_loc_ptr`. If this field is null, no information is returned. This area is mapped by BPXYWAST.

### **Options**

Supplied parameter.

#### **Type**

Integer.

#### **Length**

Fullword.

The name of a fullword that contains the wait options for this invocation of the wait-extension service.

If the `function_code` is `#wait3` or `#wait4`, the possible values are the same as the `Options` parameter for the wait (BPX1WAT, BPX4WAT) service.

If the `function_code` is `#waitid`, this parameter is used to specify which state changes to wait for. It is formed by ORing together one or more of the following flags:

#### **WEXITED**

Wait for child processes that have exited.

#### **WSTOPPED**

Return status for any child that has stopped upon receipt of a signal.

#### **WCONTINUED**

Return status for any child that has stopped and has been continued.

#### **WNOHANG**

Return immediately if there are no children to wait for.

#### **WNOWAIT**

Keep the process whose status is returned in the `info_area_ptr` parameter in a waitable state. This does not affect the state of the process; the process may be waited for again after this call completes.

These constants are defined in BPXYCONS.

**Info\_area\_ptr**

Supplied parameter.

**Type**

Pointer.

**Length**

Fullword (doubleword).

If Function\_code is #wait3 or #wait4, info\_area\_ptr is the name of a fullword (doubleword) that contains the address of a rusage structure. If this field is null, no information is returned. The rusage structure is defined in macro BPXYRLIM.

If Function\_code is #waitid, Info\_area\_ptr is the name of a fullword (doubleword) that contains the address of a siginfo\_t structure. If the function returns because a child process was found that satisfied the conditions that were indicated by the arguments Idtype and Options, the structure that is pointed to by info\_area\_ptr is completed by the system with the status of the process. If this field is null, no information is returned. The siginfo\_t structure type is defined in macro BPXYSINF.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the BPXWTE service returns -1 if it is not successful.

If it is successful and the Function\_code is #waitid, the wait-extension service returns a value of zero.

If it is successful and the Function\_code is #wait3 or #wait4, the wait-extension service returns the process id of the child that status is being reported for. If WNOHANG was specified and status is not available for any children that are specified by the Id, the wait-extension service returns a value of zero.

**Return\_code**

Returned parameter

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the wait-extension service stores the return code. The wait-extension service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The wait-extension service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>ECHILD</b>	The calling process has no existing unwaited-for child processes.
<b>EFAULT</b>	The address of a returned parameter is incorrect. The following reason codes can accompany the return code: JrBadExitStatusAddr, JrBadSiginfoAddr, or JrBadRusageAddr.
<b>EINTR</b>	The function was interrupted because the calling process received a signal.
<b>EINVAL</b>	An incorrect Option, Idtype, or Function_code was specified. The following reason codes can accompany the return code: JrBadOptions, JrBadIdType, or JrBadEntryCode.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the wait-extension service stores the reason code. The wait-extension service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. When the `siginfo_t` structure is returned, the following applies:

- **si\_signo** is always set to SIGCHLD.
- **si\_errno** is always set to 0.
- **si\_code** is set to CLD\_EXITED, CLD\_KILLED, CLD\_DUMPED, CLD\_TRAPPED, CLD\_STOPPED, or CLD\_CONTINUED. The CLD\_ constants are defined in macro BPXYSIGH.
- **si\_pid** is set to the process ID of the child that status is being returned for.
- **si\_uid** is set to the user ID of the child that status is being returned for.
- **si\_addr** is set to the faulting instruction if the child process terminated because of a SIGILL, SIGFPE, or SIGSEGV signal; otherwise, `si_addr` is set to 0.
- **si\_status** is set to the child's exit status. The exit status is mapped by macro BPXYWAST.
- **si\_band** is always set to 0.

2. If the Options field is 0, the wait-extension service waits for processes that have exited.

**Characteristics and restrictions**

None.

**Examples**

See “BPX1WTE (wait extension) example” on page 1310.

## w\_getipc (BPX1GET, BPX4GET) – Query interprocess communications

---

**Function**

The `w_getipc` service queries shared memory, messages, semaphores, and map service objects for the next or specified member to which the invoker has read access.

**Requirements**

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1GET):</b>	31-bit
<b>AMODE (BPX4GET):</b>	64-bit
<b>ASC mode:</b>	Primary mode

**Operation****Environment****Interrupt status:**

Enabled for interrupts

**Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1GET, (Ipc_Token | Ipc_Member_ID,
               Buffer_Address,
               Buffer_Length,
               Command,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GET with the same parameters. Buffer\_address is a doubleword.

**Parameters****Ipc\_Token**

Supplied parameter

**Type:**

Integer

**Length:**

Word

Specifies a token that corresponds to a message queue, shared memory segment, or semaphore member ID. Zero represents the first member ID. The token to be used in the next invocation is passed back in Return\_value. Ipc\_Token is ignored when Ipc\_OVER is specified.

**Ipc\_Member\_ID**

Supplied parameter

**Type:**

Integer

**Length:**

Word

Specifies a message queue ID, semaphore ID, or shared member ID.

**Buffer\_address**

Supplied parameter

**Type:**

Address

**Length:**

Fullword (doubleword)

Address of the buffer structure defined by IPCQ. For the structure describing this buffer, see [“BPXYIPCQ — Map w\\_getipc structure” on page 1098](#).

**Buffer\_Length**

Supplied parameter

**Type:**

Address

**Length:**

Fullword

Length of the structure defined by IPCQ. Set to IPCQ#LENGTH. Field IPCQLENGTH will differ from IPCQ#LENGTH when the system call is at a different level than the included IPCQ. An error will be returned if this length is less than 4. The buffer will be filled to the lesser of IPCQ#LENGTH or the value specified here.

### Command

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

Command	Description
<b>Ipcq#ALL</b>	Retrieve next shared memory, message and semaphore member.
<b>Ipcq#MSG</b>	Retrieve next message member.
<b>Ipcq#SEM</b>	Retrieve next semaphore member.
<b>Ipcq#SHM</b>	Retrieve next shared memory member.
<b>Ipcq#OVER</b>	Overview of system variables. Ignores the value of the first operand (Ipc_Token).
<b>Ipcq#MAP</b>	Retrieve mapped memory objects.

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the w\_getipc service returns the next Ipc\_Token (a negative number), 0, or -1 (error). If Ipc\_Token is specified, 0 indicates end of file. If Ipc\_Member\_ID is specified, 0 indicates success.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the w\_getipc service stores the return code. The w\_getipc service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The w\_getipc service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EACCES</b>	Operation permission (read) is denied to the calling process for the Ipc_Member_ID specified. The following reason code can accompany the return code: JRIpcDenied.

**Return\_code****Explanation****EINVAL**

The Ipc\_Member\_ID is not valid for the command specified:

- The Command parameter is not a valid command.
- The buffer pointer was zero or the buffer length was less than 4.

The following reason codes can accompany the return code:  
JRBuffTooSmall, JRipcBadID, or JRBadEntryCode.

**EFAULT**

An input parameter specified an address that caused the callable service to program check. The following reason code can accompany the return code:  
JRBadAddress.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the w\_getipc service stores the reason code. The w\_getipc service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. With Ipc\_Token, return\_values should be tested for 0 (end of file) or -1 (error). Other values are negative and will be the next Ipc\_Token.
2. With Ipc\_Member\_ID, return\_values should be tested for -1 (error).
3. A member's accessibility can change if the permissions are changed.
4. A given Ipc\_Token may not always retrieve the same member.
5. If a specific member is desired and has been found using Ipc\_Token, subsequent requests may place it at that token or later (never earlier).
6. The Ipc\_BINSEM, Ipc\_MEGA, Ipc\_RcvTypePID, Ipc\_SndTypePID, Ipc\_PLO1, and Ipc\_PLO2 bits in the S\_MODE field in IpcqIpcp show the values that were requested on the original get request.
7. The Ipc\_PLOinUse bit in the S\_MODE field in IpcqIpcp shows actual usage of the PLO (Perform Lock Operation) instruction for serialization.
8. When the message queue serialization uses latches, all activity is stopped for the duration of the w\_getipc request, and the values that are returned show a snapshot in time.
9. When the message queue serialization uses the PLO instruction (see Ipc\_PLOinUse), msgsnd and msgrcv activity continues during the w\_getipc request. This can cause misleading results. For example, while the w\_getipc service is counting messages on the queue, elements can be added and removed, causing a number that is too high or too low. In the same way, with the msgrcv and msgsnd waiters, a waiter's PID could appear twice in the list.
10. When the Ipcq#MAP command is specified, the w\_getipc service returns information about a map service object each time it is called. It also returns a token, which the caller can use on the next call to provide information for the system to find the next map service object.

With Ipcq#MAP, the w\_getipc service may, and likely will, return information about a particular map service object multiple times – once for each process that is using it. If the caller wishes to provide summary information, it is the caller's responsibility to associate the responses for the same map service object with each other. This can be done using the creator PID, because any one process can create only one object, and descendants of that process cannot create a map service object at all. Map service objects are inherited from the parent process.



The following information is returned for Ipcq#MAP:

- The creating process's PID. You can use this information to tie together the data returned from other `w_getipc` calls.
- The PID of a process that is using this object.
- The UID of a process that is using this object.
- The GID of a process that is using this object.
- The shutdown indicator.
- The size of the blocks.
- The number of blocks in the map area.
- The number of blocks in use.
- The number of blocks mapped by this process.

Some of these fields, which are always the same for all processes that are using a particular map service object at any one moment, may differ from one call to another. This is because they may have changed since the `w_getipc` call for an earlier process. The shutdown indicator, the number of blocks in the map area, and the number of blocks in use, in particular, may show this behavior.

## Related services

- [“shmget \(BPX1MGT, BPX4MGT\) — Create/find a shared memory segment” on page 827](#)
- [“msgget \(BPX1QGT, BPX4QGT\) — Create or find a message queue” on page 458](#)
- [“semget \(BPX1SGT, BPX4SGT\) — Create or find a set of semaphores” on page 718](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GET \(w\\_getipc\) example” on page 1242](#).

# w\_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems

---

## Function

The `w_getmntent` callable service gets information about mounted file systems.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1GMN)</b>	31-bit
<b>AMODE (BPX4GMN)</b>	64-bit
<b>ASC mode</b>	Primary address space control (ASC) mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked

**Operation****Environment****Control parameters**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1GMN,(Buffer_length,  
              Buffer,  
              Return_value,  
              Return_code,  
              Reason_code)
```

AMODE 64 callers use BPX4GMN with the same parameters.

**Parameters****Buffer\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword that contains the size of the specified buffer.

**Buffer**

Parameter supplied and returned.

**Type**

Structure

**Length**

Specified by the Buffer\_length parameter.

The name of the buffer where the information about the mount entries is stored. The area consists of a header followed by a series of entries describing the file systems, all of which are mapped by BPXYMNTE. For information about the content of this area, see [“BPXYMNTE — Map response and element structure of w\\_getmntent” on page 1104](#).

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the w\_getmntent service returns the number of mount entries that were written to the buffer, or -1 if unsuccessful. A 0 indicates that no more mount entries were found.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the w\_getmntent service stores the return code. The w\_getmntent service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return](#)

*codes (errno)* in *z/OS UNIX System Services Messages and Codes* The `w_getmntent` service can return one of the following values in the `Return_code` parameter:

Return_code	Explanation
<b>EINVAL</b>	Parameter error; for example, the buffer is too short to hold one entry, or the mount header portion of the buffer was not cleared before the first call. The following reason codes can accompany the return code: <code>JRBufTooSmall</code> , <code>JRInvalidCursor</code> , <code>JRInvalidParms</code> , and <code>JRFilesysNotThere</code> .

#### Reason\_code

Returned parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword in which the `w_getmntent` service stores the reason code. The `w_getmntent` service returns `Reason_code` only if `Return_value` is `-1`. `Reason_code` further qualifies the `Return_code` value. For a list of reason codes, see *Reason codes* in *z/OS UNIX System Services Messages and Codes*.

### Usage notes

1. Except for the situation that is described in Usage Note 5 or in Usage Note 7 before a program calls `w_getmntent` for the first time, the header part of the buffer must be cleared to zeros. For information about the format and length of this header, refer to [“BPXYMNTE — Map response and element structure of w\\_getmntent”](#) on page 1104.
2. If more than one call is made to `w_getmntent`, use the same buffer on each call, because part of the information that is returned in the buffer tells the file system where to continue retrieving its information.
3. The `w_getmntent` call normally returns information about as many file systems as are mounted, or as many as fit in the passed buffer. The number of entries that are contained in the buffer is returned. The caller must have a buffer large enough to receive information about at least a single mount entry with each call. If a zero-length buffer is passed, no information is returned, but the return value contains the total number of mounted file systems. This value could then be used to get enough storage to retrieve information about all these file systems in one additional call.

If no parameter was specified when the file system was mounted, **MNTENTPARMLEN** and **MNTENTPARMOFFSET** are each zero. If a parameter was specified, its address is the sum of the address of `MNTE` and the contents of **MNTENTPARMOFFSET**.

If an entry together with its mount parameter does not fit in the buffer, the entry is returned without the mount parameter. In this case, **MNTENTPARMLEN** contains the length of the mount parameter, and **MNTENTPARMOFFSET** is zero. To ensure that at least one entry, including the mount parameter, is returned, you should allocate space for at least two entries.

A device number of `x40000000` can be provided to indicate that the returned path name should be up to 64 bytes of the mount point path name at the time of the mount.

4. You can also retrieve all mount entries by setting up a loop that continues to call `w_getmntent` until a return value of either `-1` (in an error) or `0` (no more entries found) is returned.
5. Information about a specific file system can be obtained if the device number of that file system is known. In this case, the device number can be filled into the header of the buffer along with the eye catcher for the buffer, and the `w_getmntent` call returns a single entry with information about that file system. Or a device number value of `x40000000` can be used to indicate that the returned path name should be up to 64 bytes of the mount point path name at the time of the mount.
6. **MNTENTMOUNTPOINT** may be returned empty for a file system where **MNTENTPATHLEN** is zero and **MNTENTMOUNTPOINT** contains a null character as the first character. Reasons for this include:

- The caller of `w_getmntent` lacks search authorization to one or more of the directories in the mount point. The lack of search authorization might be caused by permission bits, ACLs, and the FSACCESS class checks for each file system that is traversed.
  - The file system is being mounted asynchronously.
  - The file system is quiesced.
7. If the caller of `w_getmntent` is requesting the additional information that is available in the expanded MNT2 data structure, MNT2, the caller must construct the buffer according to the following rules:
    - a. The buffer must be an appropriate size to hold the additional data that will be returned with the MNT2 version of the control block.
    - b. The eye catcher in the MNT2 header must be filled in with the MNT2 value.
    - c. The **bodylength** field, also in the header, must be set to the length of the MNT2 body.
  8. If an entry together with its system list does not fit in the buffer, the entry is returned without the system list. In this case, the **MNTENTSYSLISTOFFSET** is zero, and **MNTENTSYSLISTLENGTH** contains the length of the system list.
  9. When an aggregate name is present for a file system, it is included in the output if there is room for it. The offset field is set to the offset of the name from the beginning of this mount entry, and the length field is set to the length of the name. If the offset is zero and the length is nonzero, this indicates that there is an aggregate name, but there was not enough space left in the output buffer to hold it. In this case, the length field tells the program how much more space is needed.
 

Aggregate names are present for zFS file systems. They may be up to 44 characters long, and are returned in a string that is terminated by a null character. The returned length does not include the null terminator byte.
  10. A value returned in the **MNTENTROSECLABEL** (read-only security label) indicates that the file system is protected with that security label. The absence of a value in this field indicates only that a read-only security label is not in effect for that file system. It does not mean that the file system contents are not protected with security labels.
  11. Be aware that the size of the mount table could change (for instance, due to automount activity) in the interval between successive `w_getmntent` calls.
  12. For zFS file systems, the field name that contains the quiesce status of a file system depends on whether the zFS filesystem is mounted RWSHARE or read-only mode.
    - When the file system is mounted RWSHARE or in read-only mode, the quiesce status is known and controlled by zFS. Thus, the quiesce status is returned in the MNTENTPFSEXPSTATUS field.
    - When the file system is mounted NORWSHARE, the serialization of I/O operations is controlled by z/OS UNIX. The quiesce status is returned in the MNTENTSTATUS field (MNTENTQUIESCED bit).

For more information about how zFS handles quiesce conditions, see the usage notes in [zfsadm quiesce](#) in *z/OS File System Administration*.

## Related services

- [“mount \(BPX1MNT\) — Make a file system available ” on page 444](#)
- [“umount \(BPX1UMT, BPX4UMT\) — Remove a virtual file system ” on page 961](#)

## Characteristics and restrictions

None.

## Examples

See [“BPX1GMN \(w\\_getmntent\) example” on page 1246](#).

## w\_getpsent (BPX1GPS) – Get process data

---

### Function

The w\_getpsent callable service provides data describing the status of a process. This data includes, but is not limited to, running time, user IDs (UIDs), groups IDs (GIDs), and invocation parameters. Data is returned for the processes that the caller can access.

**Note:** There is no 64-bit version of the w\_getpsent callable service. To get equivalent function, use “\_\_getthent (BPX1GTH, BPX4GTH) – Get thread data” on page 325 in 64-bit mode.

### Requirements

Operation	Environment
<b>Authorization:</b>	Problem program or supervisor state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE:</b>	31-bit
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	No latches should be held
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1GPS, (Process_token,
               Buffer_length,
               Buffer_address,
               Return_value,
               Return_code,
               Reason_code)
```

### Parameters

#### Process\_token

Returned parameter

##### Type:

Integer

##### Length:

Fullword

The name of the fullword containing the process token that identifies the relative position of a process in the system. Zero represents the first process in the system.

#### Buffer\_length

Supplied parameter

##### Type:

Integer

##### Length:

Fullword

The name of the fullword containing the value PGPS#LENGTH.

## w\_getpsent (BPX1GPS)

### Buffer\_address

Supplied parameter

#### Type:

Address

#### Length:

Fullword

The name of the fullword containing the address of the buffer. For the mapping of these options, see “BPXYPGPS — Map the response structure for w\_getpsent ” on page 1116. Several fields in this buffer should be initialized:

PGPSCONTTYBLEN	Length of PGPSCONTTYBUF
PGPSCONTTYPTR	Address of PGPSCONTTYBUF (Len≠0)
PGPSPATHBLEN	Length of PGSPATHBUF
PGPSPATHPTR	Address of PGSPATHBUF (Len≠0)
PGPSCMDBLEN	Length of PGPMDBUF
PGPSCMDPTR	Address of PGPMDBUF (Len≠0)

### Return\_value

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the w\_getpsent service returns the process token or 0 if the request is successful, or -1 if it is not successful.

Value	Explanation
<b>Process Token</b>	The process token of the next logical process in the system.
<b>0</b>	End of file. There are no active processes at or following the requested process which the user is allowed access.
<b>-1</b>	Error. See Return_code for an explanation.

### Return\_code

Returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword in which the w\_getpsent service stores the return code. The w\_getpsent service returns Return\_code only if Return\_value is -1. For a list of return code values, see Return codes (errno) in *z/OS UNIX System Services Messages and Codes*. The w\_getpsent service can return one of the following values in the Return\_code parameter:

Value	Explanation
<b>EFAULT</b>	An input parameter contained the address of storage where the invoker is not authorized.
<b>EINVAL</b>	The process_token is not in the valid range.

### Reason\_code

Returned parameter

#### Type:

Integer

**Length:**

Fullword

The name of a fullword in which the w\_getpsent service stores the reason code. The w\_getpsent service returns Reason\_code only if Return\_value is -1 Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. Only those processes are returned for which RACF allows the user access based on its EUID, RUID, or SUID.
2. The normal user starts with Process\_token at zero, and continues calling BPX1GPS with the process token returned in Return\_value until the value of 0, end of file, is reached.
3. PGPSSTARTTIME is in seconds since the Epoch (00:00:00 on 1 January 1970).
4. PGPSUSERTIME and PGPSYSTIME are task-elapsed times in 1/100ths of seconds.
5. The CONTTY, PATH, and CMD input fields are initialized by the BPXYPGPS macro when it is expanded in the program CSECT for a non-reentrant program.
6. If Buffer\_length does not match that used by the callable service, the task sets PGPSLENERR on. This can reflect a change in BPXYPGPS caused by the addition of functions in later releases. This could be intentional: data is returned up to the length specified in Buffer\_length. If the length specified is less than the offset of PGPSCONTTYBLN, BPX1GPS treats the request as if the three BLN fields were zero.
7. PGPSYSTIME reports the system CPU time consumed for the address space that the process is running in. When only one process is running in the address space, this time represents the accumulated system CPU time for that process. However, when more than one process is running in an address space, the information that is returned is actually the accumulated system CPU time consumed by all of the work running in the address space.

**Characteristics and restrictions**

None.

**Examples**

The following example starts with the first process (relative process zero) and reports the status for all processes for which the invoker is allowed access (by the security access facility).

The program is reentrant and should be link-edited with RENT in the IEWL PARM.

```

BOOKSAM4 CSECT ,                Reentrant linkage
BOOKSAM4 AMODE 31
BOOKSAM4 RMODE ANY
        USING *,R15             Program addressability
@BEGIN0 B @BEGIN1               Branch around program header
        DROP R15
        DC C'Sequential w_getpsent'
        DS 0H
@BEGIN1 STM R14,12,12(13)       Save caller's registers
        LR R2,13                Hold address of caller's area
        LR R3,R1                Hold parameter register
        LR R12,R15              R12 program base register
        USING @BEGIN0,12        Program addressability
        L R0,@SIZEDAT           Size this program's dynamic area
        GETMAIN RU,LV=(0)        Getmain dynamic storage
        LR R13,R1               R13 -> this program's dynamic/save
        USING @DYNAM,13         Dynamic addressability
        ST R2,@BACK             Save caller's save area pointer
        ST 13,8(,R2)            Give caller out save area
        LR R1,R3                Restore parameter register
@BEGIN2 EQU * * * * * * *      End of the entry linkage code
        SPACE
        MVC WTOHEAD,WTOCONS     Initialize WTO line
        MVI DOT,C'.'
```

## w\_getpsent (BPX1GPS)

* If BPX1GPS has been link-edited with this program, the V-CON will be		
* resolved; if not, BPX1GPS must be loaded. In either case, the address		
* of the module is stored.		
	ICM	R0,B'1111',GPSVCON BPX1GPS address if link edited
	BNZ	STGPSEP Branch to store GPS entry point
	LOAD	EP=BPX1GPS Load w_getpsent stub
STGPSEP	ST	R0,GPSENTRY Store BPX1GPS entry point
* Initialize the variables and enter the loop.		
	XC	PROCTOKEN,PROCTOKEN Start with 1st process
	MVC	PGPSCONTTYBLEN,=A(L'PGPSCONTTYBUF) Controlling TTY
	LA	R2,PGPSCONTTYBUF
	ST	R2,PGPSCONTTYPTR
	MVC	PGPSPATHBLEN,=A(L'PGPSPATHBUF) Path name
	LA	R2,PGPSPATHBUF
	ST	R2,PGPSPATHPTR
	MVC	PGPSCMDBLEN,=A(L'PGPSCMDBUF) Command
	LA	R2,PGPSCMDBUF
	ST	R2,PGPSCMDPTR
	LA	R2,PGPS Address of PGPS buffer
	ST	R2,PGPSA
	SPACE	,
GETPS	L	R15,GPSENTRY Address of BPX1GPS load module
	CALL	(15), Get process data
		(PROCTOKEN, Relative process token
		PGPSL, Length of buffer
		PGPSA, Buffer, mapped by BPXYPGPS
		RETVAL, Return value (next, eof or error)
		RETCODE, Return code
		RSNCODE, Reason code
		VL,MF=(E,PLIST)
	SPACE	, * * * * * Test for end of file
	ICM	R2,B'1111',RETVAL Load return value, set CCode
	BZ	RETURN 0 is end of file
	BL	RETURNRC -1 is error
	ST	R2,PROCTOKEN Store the next process token
	SPACE	, * * * * * Initialize WTO area & message
	MVI	XPID,C' ' Blank variable portion of line
	MVC	XPID+1(WTO#BLANK-1),XPID
* Convert the process ID to printable hex.		
	L	R8,PGPSPID R8 = process ID
	LA	R9,XPID To be placed at message start
	LA	R15,8 8 nibbles to convert (4 bytes)
	LA	R10,9 For 0-9 / A-F compare
NIBBLE	LR	R11,R8 Target bits in 0-3
	SRL	R11,28 Bits 0-3 to 28-31
	SLL	R8,4 Drop bits 0-3 off end
	CLR	R11,R10 Are 4 bits 0-9 or A-F
	BC	B'0010',AF Branch if A-F
	LA	R11,57(,R11) Add for 0-9 (57+183=240 or F0)
AF	LA	R11,183(,R11) Add for 0-F (183+10=193 or C1)
	STC	R11,0(,R9) Store to results location
	LA	R9,1(,R9) Increment R9 to next location
	BCT	R15,NIBBLE Decrement half byte counter, loop
* Go after the state of the process		
	MVI	THREAD,C'1' Assume single task thread
	TM	PGPSSTATUS1,PGPSMULTTHREAD if multithread process
	BZ	NOTMULT
	MVI	THREAD,C'M'
NOTMULT	TM	PGPSSTATUS1,PGPSPTHREAD if pthread_create task(s)
	BZ	NOTIPT
	MVI	THREAD,C'H'
NOTIPT	MVC	STATE,PGPSSTATUS3 Z, W, X, S, C, F, K, R
	TM	PGPSSTATUS0,PGPSSWAP if swapped out
	BZ	NOTSWAP
	MVC	SWAPA,=CL4'SWAP'
NOTSWAP	TM	PGPSSTATUS1,PGPSSSTOPPED if stopped
	BZ	NOTSTOP
	MVC	STOPA,=CL4'STOP'
NOTSTOP	TM	PGPSSTATUS1,PGPSTRACE if ptrace
	BZ	NOTTRAC
	MVC	TRACA,=CL4'TRAC'
NOTTRAC	EQU	*
	SPACE	, * * * * * Display message to operator
	WTO	MF=(E,WTOAREA) Write to Operator
	SPACE	, * * * * * Loop back
	B	GETPS for the next Process data
	SPACE	,
* * * * *		
RETURN	XR	R15,R15 Zero return code
RETURNRC	L	R0,@SIZEDAT Size this program's dynamic area
	LR	R1,13 R1 -> this program's dynamic area
	L	13,@BACK R2 -> caller's save area



```

DROP 13
FREEMAIN RU,LV=(0),A=(1)
L R14,12(,13) Restore caller's R14
LM R0,12,20(13) Restore caller's R0-R12
BSM 0,R14 Branch back to caller
@SIZEDAT DC A(@ENDYN-@DYNAM) Size of dynamic storage
SPACE , * * * * * * * * * * Program constants * * * * *
PGPSL DC A(PGPS#LENGTH) Length of process data buffer
WXTRN BPX1GPS Weak to allow link edit or not
GPSVCON DC V(BPX1GPS) Get Process data module
WTOCONS DS 0CL8 Constant value for WTOHEAD
DC AL2(WTO#LENGTH) Length of area
DC AL2(0) WTO flags
DC CL4'PID=' Process ID =
SPACE , * * * * * * * * * * Dynamic storage variables
@DYNAM DSECT ,
@SAVE00 DS 0D Standard save area - 72 Bytes
DS A
@BACK DS A Backwards savearea pointer
@FORWARD DS A Forwards savearea pointer
DS 15A Regs 14,15,0-12
SPACE ,
WTOAREA DS 0F WTO message
WTOHEAD DS CL8 Mapped by WTOCONS
XPID DS CL8 Hex of process ID
DS CL1
THREAD DS CL1 1, M or H
DS CL1
STATE DS CL1 Z, W, X, S, C, F, K, R
DS CL1
SWAPA DS CL4 SWAP or blank
DS CL1
STOPA DS CL4 STOP or blank
DS CL1
TRACA DS CL4 TRAC or blank
WTO#BLANK EQU *-XPID Length to blank
DOT DS CL1
WTO#LENGTH EQU *-WTOAREA Length of WTO area
SPACE ,
GPSENTRY DS A Address of BPX1GPS
PROCTOKEN DS F Relative process token
PLIST DS 6A Calling parameter list
RETVAL DS F Return value - next PROCTOKEN
RETCODE DS F Return code
RSNCODE DS F Reason code
SPACE ,
PGPSA DC A(PGPS) ->Process data buffer
BPXYPGS DSECT=NO, Place in current dsect +
VARLEN=(0,0,0) ConTty=0,Path=0,Cmd=0
@ENDYN EQU * End of dynamic storage
SPACE 3 * * * * * * * * * * Register equates * * * * *
R0 EQU 0
R1 EQU 1 Parameter list pointer
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
* 12 Program base register
* 13 Savearea & dynamic storage base
R14 EQU 14 Return address
R15 EQU 15 Branch location
SPACE ,
END

```

## w\_ioctl (BPX1IOC, BPX4IOC) – Control I/O

### Function

The `w_ioctl` callable service conveys a command to a device. The specific actions that are specified by the `w_ioctl` callable service vary by device and physical file system, and are defined by the device driver or physical file system.

The SIOCGPARTNERINFO ioctl provides an interface for an application to retrieve information about its partner, including connection routing information, the user ID of the partner, or both. For more information, see [Type 119 SMF records in z/OS Communications Server: IP Programmer's Guide and Reference](#).

The SIOCSPARTNERINFO ioctl provides an interface for an application to set up the environment that is required to retrieve the user ID of its partner by using the SIOCGPARTNERINFO ioctl. Issuing the SIOCSPARTNERINFO ioctl before the SIOCGPARTNERINFO ioctl can provide better performance, potentially eliminating wait time when issuing the SIOCGPARTNERINFO ioctl. Refer to [Type 119 SMF records in z/OS Communications Server: IP Programmer's Guide and Reference](#) for details.

## Requirements

Operation	Environment
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task or SRB (AF_INET/AF_INET6 socket support only)
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1IOC)</b>	31-bit task or SRB mode
<b>AMODE (BPX4IOC)</b>	64-bit task mode only
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

## Format

The syntax format is as follows:

```
CALL BPX1IOC, (File_descriptor,
               Command,
               Argument_length,
               Argument,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4IOC with the same parameters.

## Parameters

### File\_descriptor

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the file descriptor of an open file or a socket descriptor.

### Command

Supplied parameter.

#### Type

Integer

**Length**

Fullword

The name of a fullword that contains the ioctl command that is to be passed to the device driver or physical file system.

See [“BPXYIOCC — Ioctl command definitions”](#) on page 1085 for a complete list of the commands that are supported.

**Argument\_length**

Supplied parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword containing the length of the argument. The length of the argument is specified as an integer value in the range 0–51 200.

**Argument**

Parameter supplied and returned.

**Type**

Defined by the device driver or physical file system.

**Character set**

No restriction.

**Length**

Specified by the Argument\_length parameter.

Specifies the name of a buffer, of length Argument\_Length, containing the argument to be passed to the device driver or physical file system.

For extended attributes, the area is mapped by the attx structure in the same BPXYATT mapping. Storage for the attribute value must be contiguous with the attx structure and the attributes\_length argument is the length of the attx structure plus the length of the attribute value.

**Return\_value**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the w\_ioctl service returns one of the following values:

Return_value	Explanation
0	Request was successful. For the <b>getfacl</b> command, return_value contains the ACL length if the request is successful.
-1	Request was not successful.
1	The SIOCSECENVR ioctl with the SIOC#GETENVR argument was issued and the buffer size that is specified with the SECO_BUFFERLEN argument was zero or was not large enough to contain the security object. (See usage note <a href="#">“19”</a> on page 1017.)

**Return\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the w\_ioctl service stores the return code. The w\_ioctl service returns the return code only if the return value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The w\_ioctl service returns one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>E2BIG</b>	The argument_length passed on a SetfACL or GetfACL request was not large enough to contain even the minimum amount of data. The specified size must be large enough to hold a RACL_Edit, followed by an FACL and as many FACL_Entry entries as needed.
<b>EACCES</b>	Permission is denied. The following reason codes can accompany the return code: JrIpcDenied.
<b>EALREADY</b>	An attempt was made to unregister a file that is not registered.
<b>EBADF</b>	The fildes parameter is not a valid file or socket descriptor. The following reason code can accompany the return code: JrFileNotOpen.
<b>EFAULT</b>	The address is incorrect. The following reason codes can accompany the return code: JrReadUserStorageFailed and JrWriteUserStorageFailed.
<b>EIBMBADTCPNAME</b>	The command passed was IOCC#DIRIOCTL and the stack name was not found attached to this socket. The specific error is determined by the reason code that accompanies this return code: <p><b>JrNoCINET</b> Common INET is not configured or this is not a socket and the name did not match the PFS name. This error might not be critical to the application because the embedded ioctl can be sent directly to the one and only stack or PFS as a regular ioctl.</p> <p><b>JrCINETBadName</b> CINET is configured and this name does not match any stack.</p> <p><b>JrCINETNotAttached</b> CINET is configured and this name matches a stack, but that stack is not attached to this socket.</p>
<b>EINVAL</b>	One of the following occurred: <ul style="list-style-type: none"> <li>• The w_ioctl service specified an incorrect length for the argument. The correct argument length range is 0-51 200.</li> <li>• An invalid command was encountered.</li> <li>• An invalid message queue was specified when using command _IOCC_REGFILEINT.</li> </ul> <p>The following reason codes can accompany the return code: JRInvIoctlCmd, JrNotSupportedForFileType, JrFileNotOpen, JrBadSubField, JrIpcBadId.</p>
<b>EINVAL</b>	The extended attribute operation is invalid due to the values that are passed. The input length for the ATTX parameter might be insufficient or the subcommand AttxSubCmd is invalid. The following reason codes can accompany the return code: JRInvMinArgLenAttx, JRInvArgLenAttx, JRInvAttxCmd.

Return_code	Explanation
<b>EIO</b>	<p>One of the following issues occurred:</p> <ul style="list-style-type: none"> <li>• The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.</li> <li>• A network or transport failure occurred.</li> </ul> <p>The following reason codes can accompany the return code: JRSingleTDRegd, JRPrevSockError.</p>
<b>EMVSPARM</b>	Incorrect parameters were passed to the service. The following reason codes scan accompany the return code: JRNoStorage and JRInvParmLength.
<b>ENOATTR</b>	For the get function, the specified attribute is not set. For list, no attributes are set. The following reason codes can accompany the return code: JrAttxUnavail, JRAttxNoSup, and JrAttxName.
<b>ENOBUFFS</b>	Insufficient buffer space available. The following reason code can accompany the return code: JrNoArea.
<b>ENODEV</b>	The device is incorrect. The function is not supported by the device driver. The following reason code can accompany the return code: JRFuncNotSupported.
<b>ENOTTY</b>	The w_ioctl service specified an incorrect file descriptor. The file type was not character special. The following reason codes can accompany the return code: JRNotSupportedForFileType.
<b>ERANGE</b>	The buffer length that is provided in AttxValLn is not large enough to return the attribute or list. The following reason code can accompany the return code: JrInvAttxValLen.

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the w\_ioctl service stores the reason code. The w\_ioctl service returns the return code only if the return value is -1. The reason code further qualifies the return code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Extended attribute notes for w\_ioctl**

1. The extended attributes functions are invoked by using the IOCTL command code for **extattr**, IOC#ATTR, which is in BPXYIOCC. Use the ATTX control block in the BPXYATT macro as the Argument parameter.
2. Only the Attx#scGet and Attx#scList functions are handled by this function. The ATTX mapping is variable length, with the **AttxVal** field being a char(\*) value.
  - For the get function, the caller fills in AttxName and AttxVal by specifying the attribute name and length. The AttxValLn field can be set to zero to get the size needed. It can also be set to the size of the space available to contain the attribute's value. On return, if AttxValLn was zero on input then Return\_value is set to the AttxValLn size needed to return the attribute. Otherwise, if the attribute's value fits within AttxValLn the value is returned starting at AttxVal and Return\_Value is set to the length of the value. If the attribute's value does not fit, then Return\_value is set to -1 and Return\_Code is set to ERANGE.

- For the list function, the caller's values of AttxName and AttxNameLn are not used. The AttxVal field refers to the space to receive the list. The AttxValLn field can be set to zero in which case Return\_Value is set to the size required for the current list. If AttxValLn is set to nonzero, it should be set to the size of the area to contain the returned list of attribute names. On return, if the list fits within the provided space, the list is placed in AttxVal and Return\_Value is set to the length of the returned list. If the list does not fit, then Return\_Value is set to -1 and Return\_Code is set to ERANGE.

3. The get function extracts the attribute information that also exists in BPXYSTAT. The following table provides a correspondence between the information in BPXYATT and BPXYSTAT.

<i>Table 33. Correspondence between BPXYATT and BPXYSTAT</i>			
<b>Xattr name</b>	<b>Existing SET attribute in BPXYATT</b>	<b>Existing GET attribute in BPXYSTAT</b>	<b>Restrictions and comments</b>
trusted.apfauth	attapfauth	ST_APFAUTH	Set and remove functions that are limited according to "general flags" restrictions.
trusted.sharelib	attsharelib	ST_SHARELIB	Set and remove functions that are limited according to "general flags" restrictions.
trusted progctl	attprogctl	ST_PROGCTL	Set and remove functions that are limited according to "general flags" restrictions.
system.noshareas	attnoshareas	ST_NOSHAREAS	Set and remove functions that are limited according to "general flags" restrictions.
system.filefmt	attfilefmt	ST_FILEFMT	None.
system.filetag	attfiletag	ST_FILETAG	None.
system.seclabel	attseclabel	ST_SECLABEL	None.
system.useraudit	attuseraudit	ST_USERAUDIT	None.
system.auditoraudit	attauditoraudit	ST_AUDITORAUDIT	None.
system.auditid	Not an attribute. Equivalent to field ST_AUDITID in BPXYSTAT.	ST_AUDITID	Read-only.
system.dmodelacl	Not an attribute. Equivalent to field.	ST_DMODELACL	Read-only.
system.fmodelacl	Not an attribute. Equivalent to field.	ST_FMODELACL	Read-only.
system.accessacl	Not an attribute. Equivalent to field.	ST_ACCESSACL	Read-only.
system.createtime	Not an attribute. Equivalent to field.	ST_CREATETIME	Read-only.
user.*	No equivalent.	No equivalent.	Needs specific PFS support.

## Usage notes

- See Chapter 3, "Callable services for SRB mode routines," on page 1041 for more information about programming considerations for SRB mode.

2. z/OS UNIX domain sockets support the following commands:

- FIONBIO
- FIONREAD
- FIONWRITE
- SECIGET
- SIOCATMARK
- SIOCSECEENV

3. Inet sockets pass the ioctl command to TCP/IP. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for the commands that are supported.

4. Pseudoterminals (ptys) and remote terminals (rtys) support TIOCGWINSZ and TIOCSWINSZ to get and set the window size. Pseudoterminals also support TIOCNOTIFY begin (IOCC#PWBEGIN) and TIOCNOTIFY end (IOCC#PWEND) secure data mode.

TIOCGWINSZ and TIOCSWINSZ retrieve and store the winsize structure (BPXYWNSZ). TIOCNOTIFY sets the TIOCPKT\_PWBEGIN and TIOCPKT\_PWEND bits on the main read() when in extended packet mode.

5. The pipe file system does not support ioctl.

6. The IOCC#UPDTOFTE command updates a 100-byte state area that is associated with an Open File Table Entry (OFTE). The Open File Table Entry is created by the socket, open, and pipe functions, and are shared by child processes.

This function is intended for use by runtime libraries.

The Argument buffer contains an UPDTOFTE subcommand and the offset, length, and value of the data to be updated. Refer to the BPXYIOCC macro for the mapping of this structure.

Data that is written to or read from the state area is addressed by offset and length within the state area. The state area is initialized to all zeros when it is allocated.

Three subcommands are available:

- **IocUo#Write**

The specified data value is written to the specified offset in the state area. This subcommand also initially allocates the area and must be the first UPDTOFTE subcommand issued.

- **IocUo#Read**

The data at the specified offset in the state area is returned.

- **IocUo#CS**

This is used for a "compare and swap" type write to the state area. The specified old\_value is compared to what is in the state area at the old\_offset. If they match, the new\_value is written to the new\_offset. If they do not match, the current data at the old\_offset in the state area is returned in the old\_value along with a Return\_Value of 1. The old data and the new data do not have to be at the same offset within the state area.

All of the subcommand operations are atomic with respect to other tasks attempting to access the same OFTE state area.

7. The **Ioccc#RegFileInt** command registers interest in a file and allows the program to be notified when a change to that file occurs. Unauthorized processes are limited to 1000 registrations.

To register file interest, the user must have both read and write permissions to the message queue.

The program creates one or more IPC message queues and specifies a Queue ID on each registration, along with a message type and a user token that identifies the file to the program. These are specified in the Rfis structure in the BPXYRFIS macro. See [“BPXYRFIS — Map the register file interest structures” on page 1141](#). A Registered File Token, RfTok, is returned from the registration; this can be used later to unregister the file.

You can register files by descriptor with the `w_ioctl` service, or by path name with the `w_pioctl` service.

When a change occurs to a registered file, a message is sent on the registered IPC Message Queue. The message content is described by the `Rfim` structure in the `BPXYRFIS` macro, and contains:

- The message type specified on registration.
- The user token specified on registration.
- The type of change that occurred.

The types of file changes that generate a message are:

- File write, including truncate and `open(O_TRUNC)`.
- Any attribute change, such as a `chmod` or `chown` request.
- Renaming, removal, or unlinking of any of the file's names.
- Attempts to unmount the containing file system.

Because a registered file is implicitly unregistered when a message is sent, only one message is sent for any given registration.

A file can be explicitly unregistered with the `w_ioctl` or the `w_pioctl` service. An `Rfis` structure is passed on these calls that contains the `RfTok` that was returned when the file was registered. The file descriptor or path name that is used on the call is not important, but it must be valid. If the registered file is no longer open, and its file descriptor is therefore not readily available, you can use the `w_pioctl` service with a path name of `/`.

If you try to unregister a file that has already been implicitly or explicitly unregistered, the call fails with `EALREADY`. If you receive this return code, there might be a message waiting for you on the queue, so you should coordinate the freeing of any file-related control blocks that might be referenced when that message is read.

All file registrations are removed if the registering process terminates or issues an `exec`-type call and no messages are sent.

The queue should be created with permission bits of 600.

The queue must be large enough to accommodate the expected number of unprocessed messages. Also, the messages must be processed fast enough so that the system limit on total outstanding messages is not exceeded. Messages that cannot be queued immediately are discarded, but the fact that messages were lost is remembered. This information is communicated to the application in one of two ways:

- The `Rfim_LostMsgs` flag is set on subsequent change messages that are sent to this process until a message is successfully queued.
- The `Rfis_LostMsgs` flag is returned on the next successful registration or unregistration.

When an application is informed that messages have been lost, it must unregister all registered files, ignoring any `EALREADY` return codes. Then, it must drain the message queue, ignoring any change messages received and start over.

Program errors can also prevent messages from being delivered; for example, if a bad queue id is specified on registration. When a message cannot be delivered, a `Ctrace` entry is written for component `SYSOMVS` of type `FILE`. The trace entry contains the character string "RFIPCERR", the returned failure codes from the `msgsnd` service, the queue id used, and the message that was being sent. You can use this information during program development to diagnose simple bugs.

A registered file does not have to be open to be, or to remain, registered.

A file can be registered multiple times, and by different processes. Each registration causes a separate message when the file is changed.

Any file type can be registered, but some change events only apply to regular files. In particular, writes to a directory (that is, file creation and deletion) do not generate a change message for a registered directory.



No special authority is required to register a file. Any file that the caller has open or is allowed to make stat() calls to can be registered.

Registration and file change notification are intended for use by programs that would otherwise issue periodic stat() or fstat() calls to monitor a file's timestamps in order to detect changes to the file.

8. The **Iocctl#RegFileInt** command will fail with ENOSYS if it is invoked to register a file that resides in a sysplex-aware file system that is locally mounted.
9. For file systems that support access control lists (ACLs), you can use the following commands:

#### **GetfACL**

Retrieves information from an access control list. The Argument parameter specifies the user buffer containing the following input:

- A structure of type RACL\_EDIT, defined in IRRPCOMP, followed by
- A structure of type FACL, defined in IRRPFACL.

Upon successful return, the buffer holds the requested ACLs. The size of the buffer that is passed to BPX1IOC (specified by Argument\_length) must be big enough to hold the returned ACLs. If it is not large enough, another call is needed. The maximum number of ACL entries is 1024.

Set the RACL\_EDIT and FACL fields as follows:

#### **RACL\_EDIT\_OPTYPE**

0

#### **RACL\_EDIT\_ACLTYPE**

The type of ACL being requested (RACL\_ACCESS, for instance). You must issue separate calls for access and default ACLs.

#### **FACL\_ID**

FACL

#### **FACL\_LEN**

Size of FACL (FACL\_LENGTH)

#### **FACL\_LEN\_ENTRY**

FACL\_ENTRY\_LENGTH

#### **FACL\_VERS**

Version number (for example, X'01')

Upon successful return, FACL\_NUM\_ENTRY (offset X'12') contains the number of ACL entries that the file has of the specified type (access or default). It is up to the caller to determine whether the buffer is large enough to hold that many entries. The calculation for the amount of space needed is:

```
(length of RACL_EDIT structure) + (FACL_NUM_ENTRY × FACL_LEN_ENTRY)
```

The entries start at FACL\_ENTRIES and are mapped by FACL\_ENTRY.

#### **SetfACL**

Sets information into an access control list. There are four types of operations that you can perform on access or default ACLs:

- a. Delete a whole ACL.
- b. Add a whole ACL.
- c. Add or change individual ACL entries.
- d. Delete individual ACL entries.

The contents and length of the user buffer that is passed in the Argument parameter depend on the type of operation, as follows:

- a. Deleting a whole ACL: Only a RACL\_EDIT structure needs to be passed in the buffer and the buffer only needs to be as large as that structure. Set the RACL\_EDIT fields as follows:

**RACL\_EDIT\_OPTYPE**

RACL\_DELETE

**RACL\_EDIT\_ACLTYPE**

RACL\_ACCESS, RACL\_FILEMOD, or RACL\_DIRMOD

If the ACL is not found, the request is ignored.

- b. Adding a whole ACL: A RACL\_EDIT structure, FACL structure, and all FACL\_ENTRY blocks to be added must be passed in the Argument buffer. Argument\_length must indicate the size of the entire buffer. Set the RACL\_EDIT, FACL, and FACL\_ENTRY fields as follows:

**RACL\_EDIT\_OPTYPE**

RACL\_ADD

**RACL\_EDIT\_ACLTYPE**

RACL\_ACCESS, RACL\_FILEMOD, or RACL\_DIRMOD

**FACL\_ID**

FACL

**FACL\_LEN**

FACL\_LENGTH + (number of FACL\_ENTRYs × FACL\_ENTRY\_LENGTH). Do not include the length of RACL\_EDIT.

**FACL\_LEN\_ENTRY**

FACL\_ENTRY\_LENGTH

**FACL\_VERS**

Version number (for example, X'01')

**FACL\_NUM\_ENTRY**

Number of FACL\_ENTRY blocks in the buffer

Set the following fields for each FACL\_ENTRY, as appropriate:

**FACL\_READ**

1 (to give permission)

**FACL\_WRITE**

1 (to give permission)

**FACL\_EXECUTE**

1 (to give permission)

**FACL\_ENTRY\_TYPE**

X'01' for user or X'02' for group

**FACL\_ENTRY\_ID**

UID or GID (in decimal), based on FACL\_ENTRY\_TYPE

- c. Adding or changing individual ACL entries: A RACL\_EDIT structure, FACL structure, and all FACL\_ENTRY blocks to be added or modified must be passed in the Argument buffer. Argument\_length must indicate the size of the entire buffer. Set the RACL\_EDIT, FACL, and FACL\_ENTRY fields as follows:

**RACL\_EDIT\_OPTYPE**

RACL\_MODIFY

**RACL\_EDIT\_ACLTYPE**

RACL\_ACCESS, RACL\_FILEMOD, or RACL\_DIRMOD

**FACL\_ID**

FACL

**FACL\_LEN**

FACL\_LENGTH + (number of FACL\_ENTRYs × FACL\_ENTRY\_LENGTH). Do not include the length of RACL\_EDIT.

**FACL\_LEN\_ENTRY**

FACL\_ENTRY\_LENGTH

**FACL\_VERS**

Version number (for example, X'01')

**FACL\_NUM\_ENTRY**

Number of FACL\_ENTRY blocks in the buffer

Set the following fields for each FACL\_ENTRY, as appropriate:

**FACL\_READ**

1 (to give permission)

**FACL\_WRITE**

1 (to give permission)

**FACL\_EXECUTE**

1 (to give permission)

**FACL\_ENTRY\_TYPE**

X'01' for user or X'02' for group

**FACL\_ENTRY\_ID**

UID or GID (in decimal), based on FACL\_ENTRY\_TYPE

If the entry is not found in the existing ACL, it is added as a new entry. If the entry is found for the given user or group, it is modified with the specified permissions.

- d. Deleting individual ACL entries: A RACL\_EDIT structure, FACL structure, and all FACL\_EDIT\_ENTRY blocks to be deleted must be passed in the Argument buffer. Argument\_length must indicate the size of the entire buffer. Set the RACL\_EDIT, FACL, and FACL\_EDIT\_ENTRY fields as follows:

**RACL\_EDIT\_OPTYPE**

RACL\_MODIFY

**RACL\_EDIT\_ACLTYPE**

RACL\_ACCESS, RACL\_FILEMOD, or RACL\_DIRMOD

**FACL\_ID**

FACL

**FACL\_LEN**

FACL\_LENGTH + (number of FACL\_EDIT\_ENTRIES × FACL\_ENTRY\_LENGTH). Do not include the length of RACL\_EDIT.

**FACL\_LEN\_ENTRY**

FACL\_ENTRY\_LENGTH

**FACL\_VERS**

Version number (for example, X'01')

**FACL\_NUM\_ENTRY**

Number of FACL\_EDIT\_ENTRY blocks in the buffer

Set the following fields for each FACL\_EDIT\_ENTRY to be deleted:

**FACL\_DEL\_ENTRY**

1

**FACL\_EDIT\_TYPE**

X'01' for user or X'02' for group

If the entry is not found in the existing ACL, it is ignored. If the entry is found for the given user or group, it is deleted. You can have entries to be deleted along with entries to be added in the same buffer.

The following FACL\_ENTRY fields are useful for debugging:

**FACL\_RACF\_RETURN\_CODE**

The return code from RACF, documented in [z/OS Security Server RACF Callable Services](#). See the return and reason codes for the makeFSP service.

**FACL\_RACF\_REASON\_CODE**

The reason code from RACF, documented in *z/OS Security Server RACF Callable Services* (see the return and reason codes for the makeFSP service).

**FACL\_ERROFF**

If the problem is with an entry, this field indicates the offset into the Argument buffer where the problem occurred.

Also, refer to the usage notes for the R\_setfac1 service in *z/OS Security Server RACF Callable Services*.

10. The IOCC#DIRIOCTL (Directed Ioctl) command sends an embedded ioctl command and argument to a specified stack. The input argument for this command is the IocDirIoctl structure, from the BPXYIOCC macro ("[BPXYIOCC — Ioctl command definitions](#)" on page 1085), with the following fields:

**IocDirName**

The name of the stack.

**IocDirCmd**

The ioctl command to be sent to IocDirName.

**IocDirArgLen**

The length of IocDirArg, which follows.

**IocDirArg**

The ioctl argument to be sent to IocDirName.

The embedded ioctl is passed to the specified stack, if that stack is attached to this socket, without any examination or processing by the system. Any errors that are returned are usually returned by the stack. Directed Ioctl is not strictly restricted to socket stacks. The name should match the PFS name for the descriptor that is used.

If the embedded ioctl generates output in its argument buffer, the output is returned in the IocDirArg buffer.

A unique error can be returned by z/OS UNIX System Services for this ioctl command, EIBMBADTCPNAME, when the stack name is not found attached to this socket.

11. The IOCC#GETSTACKS (Get TCPIP Stack Names) command returns the names of all the transport stacks that are attached to a socket, and information related to those stacks. The output argument for this command is the IocStackInfo structure, from the BPXYIOCC macro ("[BPXYIOCC — Ioctl command definitions](#)" on page 1085), with the following fields:

**IocStackEntries**

The number of **IocStackName** array entries that were returned. With CINET, one or more entries might be returned, depending on how many stacks have been configured under CINET, how many are or have been active, and any stack affinity that might have been established for the socket or process.

**IocStackName**

The name of the stack.

**IocStackCINET**

Indicates that this is a CINET socket. When this bit is on, the IocStack\_IPv6\_Interfaces and IocStack\_IPv4\_Interfaces flags indicate whether the specified stack has configured interfaces of each type. Without CINET, use the SIOCGIFVERSION ioctl to obtain this information directly from the Inet stack. For more information about the SIOCGIFVERSION ioctl command, see [SIOCGIFVERSION \(determine if an IPv4 or IPv6 interface has been configured on a TCP/IP stack\)](#) in *z/OS UNIX System Services File System Interface Reference*.

**IocStack\_IPv6\_Support**

Indicates that this stack supports IPv6 protocols and sockets that are created with AF\_INET6. CINET supports IPv6 sockets over stacks that do not themselves support IPv6, as long as IPv4-mapped addresses can be used.

**IocStackTdIndex**

The CINET TdIndex for this stack. This value is used in the upper halfword of Interface Indices when CINET is configured.

**IocStack\_Active**

Indicates that this stack is active. When used with this ioctl command, this bit is usually on, because inactive stacks are not usually attached to a socket, unless the stack has recently terminated.

This ioctl is not strictly restricted to socket stacks; however, with any other type of Physical File System, all of the socket-related flags would be off.

**Tip:** You can use the PC#TdNames pfscctl command function of the pfscctl (BPX1PCT, BPX4PCT) service to obtain a complete list of all the stack names, active or inactive, that are configured under CINET.

12. The **IOCC#GRTRSELECT** (Get CINET PreRouter Selections) command returns the CINET stack that would be chosen for each of a list of destination IP addresses. This ioctl is passed an array of IP addresses, and returns for each address the CINET stack that would be chosen for that destination. This is the stack over which a connect() or sendto(), for instance, would be routed if that address were specified on the call at this time. If CINET is not configured, the socket's one and only stack is returned. The input and output argument for this command is the IocRtrSelect structure, from the BPXYIOCC macro ([“BPXYIOCC — Iocctl command definitions” on page 1085](#)), with the following fields in each array entry:

**IocRtrIpAddr**

Specifies the IP address to test. This is an IPv6 address or an IPv6-mapped IPv4 address.

**IocRtrStack**

Returns the name of the stack that would be chosen.

**IocRtrErrTest**

When equal to B'0', this indicates that there was an error with this one IP address. The following two fields are also returned: IocRtrErrno, which contains the failing return code (errno), and IocRtrRsn, which contains the failing reason code.

13. The **SIOCGIFNAMEINDEX** (Get Interface Name/Index Table) command returns the Interface Name/Index table for every stack that is attached to a socket. The output argument for this command is the If\_NameIndex structure, from the BPXYIOCC macro ([“BPXYIOCC — Iocctl command definitions” on page 1085](#)), with the following fields:

Field	Description
-------	-------------

**If\_NITotalIF**

Contains the total number of interfaces that have indices assigned on the stacks that are attached to this socket.

**If\_NIEntries**

Contains the number of interfaces that have been returned. When the total is greater than the number of entries returned, the supplied buffer was not large enough to hold all of the required information. In that case, If\_NITotalIF can be used to calculate the amount of space needed and the call can be repeated. When all the interfaces can be returned, the two values are equal.

**If\_NITable**

Contains an array of If\_NameIndexEntry structures.

Each interface is described by an If\_NameIndexEntry structure consisting of:

**If\_NIIndex**

Contains the Interface Index, as described in this topic.

**If\_NIName**

Contains the Interface Name, as a 1- to 16-byte character string, left-justified, and padded with blanks. When there is more than one stack, these names may not be unique, because the names are defined to each stack individually with their own configuration procedures.

**If\_NINameTerm**

A null character supplied to terminate the name string for the convenience of C routines.

**Tip:** To query for the total number of interfaces, you can specify an argument length of 8, just large enough for the first two fields, and the total will be returned in If\_NITotalIF, with an If\_NIEntries value of 0.

This output is similar to the output of the if\_nameindex() C/C++ function. For a CINET socket with more than one stack attached, the tables from each stack are concatenated into one output table. For a CINET socket, in general, a Transport Driver Index, TdIndex, value will be inserted into the Interface Indices to uniquely identify the interfaces. For example, with two stacks (1) TCPA, with interfaces IFA1 and IFA2, whose interface indices are 1 and 2, respectively, within TCPA, and (2) TCPB, with interfaces IFB1 and IFB2, whose interface indices are 1 and 3, respectively, within TCPB, the output of this ioctl would be something like:

```
('00010001'x, IFA1), ('00010002'x, IFA2), ('00020001'x, IFB1), ('00020003'x, IFB2)
```

The first halfword of the index value indicates which stack under CINET the interface belongs to. The second halfword contains that stack's interface index for this interface.

Without CINET, if TCPA was configured as the only stack, and it was IPv6-enabled, the output of this ioctl would be:

```
('00000001'x, IFA1), ('00000002'x, IFA2)
```

Interface indexes are used in various places in IPv6, such as for the scope\_id of the IPv6 sockaddr structure and within the in6\_pktinfo structure. In a CINET configuration, the first halfword of an interface index is used to route a call to the corresponding numbered stack. The upper halfword is cleared before the data is passed to the stack, so that one could use interface indexes of the form X'000N0000' as a way to route a call to stack number N without actually specifying an interface index to that stack. The specified stack must be attached to the current socket. The stacks under CINET are numbered in the order of the SUBFILESYSTYPE statements in the BPXPRMxx parmlib member that defined the configuration. These values can be determined from the IocStackTdIndex field of the Iocc#GetStacks ioctl, or from the order of the names returned by the PC#TdNames pfscctl.

Refer to the C/C++ functions if\_nameindex(), if\_nametoindex(), and if\_indextoname() for more information about interface names and indexes. (See *z/OS XL C/C++ Runtime Library Reference*.)

To request OSM interfaces the application must have READ authorization to the EZB.OSM.sysname.tcpname resource.

14. The SIOCGSOCKPOEATTRS and SIOCGFDPOEATTRS commands return port of entry information for multilevel security. SIOCGSOCKPOEATTRS returns port of entry attributes for a socket resource. SIOCGFDPOEATTRS returns port of entry attributes for a non-socket resource. The port of entry information that can be returned by these commands is defined in the IocPoeAttr block in [“BPXYIOCC — Ioctl command definitions”](#) on page 1085.
15. The **Iocc#DevConsole** command allows a program with appropriate privileges to suppress the message number and user ID that are normally prefixed to message BPXF024I when text that is written to /dev/console is sent to the system console. The Argument\_length must be 4 and an Argument value of Iocc#DevConSuppress (1) enables suppression of the header on future writes. An Argument value of Iocc#DevConUnSupprs (0) cancels the suppression, so future writes will contain the header.
16. The **SECIGET\_T ioctl** command returns both process-level and, if available, task-level security information of the peer for an AF\_UNIX stream-connected socket. The task-level security information is from the task that issued the connect or accept call. The security information is returned in a BPXYSECT structure. The security information is not available until accept() completes. The availability of the peer's task-level security data is determined by the task-level userID length field. If the length is zero, the peer does not have task-level security data.
17. The SIOCTIEDESTHRD ioctl command with the SIOC#TIESD argument ties or associates a descriptor with the thread of the SIOCTIEDESTHRD caller. If that task terminates before the descriptor is closed

or untied from the task, then the termination processing for the file system thread will close the descriptor.

The SIOCTIEDESTHRD ioctl command with the SIOC#UNTIESD argument unties a previously tied descriptor from a thread. SIOCTIEDESTHRD can be used on heavy-weight and medium-weight threads.

18. The FIONWRITE ioctl command returns the number of bytes that can be written to the connected peer AF\_UNIX stream socket before the socket blocks or returns an EWOULDBLOCK return code. Note that the number of bytes returned by FIONWRITE is not guaranteed unless there is serialization among the calling applications.
19. The SIOCSECENVR ioctl command sets or gets the security environment of a client that is connecting to an AF\_UNIX stream socket server. Arguments for the SIOCSECENVR ioctl are mapped by the BPXYSECO structure. (For more information about BPXYSECO, see “BPXYSECO — Map the input/output of BPX1IOC for the SIOCSECENVR request” on page 1144). A server must have appropriate privileges to issue this ioctl.
  - The SIOCSECENVR ioctl with the SIOC#SETENVR argument is for use by an AF\_UNIX stream socket server to mark the server socket as one that requires the full security environment of a connecting client to be available before a connect() will successfully complete. The connect service obtains the security environment of the connector and anchors it off of the connector's socket for use by the server. If the security environment cannot be obtained during connect processing, the connect() will fail. This ioctl is meaningful only for sockets that will become server sockets; it has no effect for all other sockets.
  - The SIOCSECENVR ioctl with the SIOC#GETENVR argument is for use by an AF\_UNIX stream socket server to copy the previously set security environment from the connector's address space to the server's address space so that it can be used as input on calls to the security product. This ioctl is only meaningful for server sockets that previously issued the SIOCSECENVR ioctl with the SIOC#SETENVR argument.

Servers must issue the SIOCSECENVR ioctl with the SIOC#GETENVR argument in a timely fashion. It should be issued immediately following the accept() call. If any read() calls are issued before the SIOC#GETENVR request, then the server will no longer be able to use a SIOC#GETENVR request to obtain the client's security environment.

Servers can specify the buffer in which to hold the client's security environment in the BPXYSECO structure. If the specified buffer is not large enough to contain the security environment or if SECO\_BUFFERLEN is zero, the service will obtain a buffer of the correct size in the server's address space and return the security environment in that buffer. Information about the buffer and the security environment is returned in the BPXYSECO structure and the return value is set to 1. The server must free this buffer when it no longer needs it.

The security environment that is returned by a SIOC#GETENVR request can be specified as input to the RACROUTE interface using the ENVRIN keyword or to the initACEE callable service using the ENVR\_in parameter.

20. The SIOCGIFCONF6 (Get IPv6 Interface Configuration) command gets the name, address, and other information about the configured IPv6 network interfaces. This is similar to the SIOCGIFCONF command for IPv4.

A Net\_IfConf6Header structure is passed as the argument of the ioctl. This structure specifies the buffer where the configuration information is to be written and is returned with the number of entries and entry length of the Net\_IfConf6Entry structures that were written to the output buffer. These structures are defined in the BPXYIOC6 macro.

If the specified buffer address and buffer length are both zero, a Query function is performed and the header is returned with the total number of entries that would be output and the length of each individual entry for the specified version. If the specified version is zero or not supported, it is replaced with the maximum supported version and the entry length returned corresponds to that version.

If a call to get information fails with either return code ERANGE or with both return code EINVAL and the Nif6h\_Version field having been changed, the call was converted into a Query function and the header has been filled. In this case, the content of the output buffer is unpredictable.

If Common INET (CINET) is configured and multiple TCP/IP stacks are attached to the socket, the output from each stack that is enabled for IPv6 is concatenated in the output buffer and the header will contain the total number of entries returned from all the stacks. The version that is returned with the Query function is the highest version supported by all the stacks.

This ioctl can be issued on an AF\_INET or AF\_INET6 socket.

To request OSM interfaces the application must have READ authorization to the EZB.OSM.sysname.tcpname resource.

21. The **Ioccc#GetPathName** and **Ioccc#GetPathNameRel** (get pathname and get relative pathname) commands return the absolute or relative path name, respectively, of the file referred to by *File\_descriptor*. The output path name is placed in the Argument buffer and is terminated by a null character. The length of the output path name is determined by scanning for the trailing null byte. The Argument buffer provided must be large enough to contain the output name and the trailing null byte or the call will fail with RC=ERANGE.
22. The SIOCGPARTNERINFO ioctl provides an interface for an application to retrieve information about its partner, including connection routing information, the user ID of the partner, or both. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for details.
23. The SIOCSPARTNERINFO ioctl provides an interface for an application to set up the environment that is required to retrieve the user ID of its partner by using the SIOCGPARTNERINFO ioctl. Issuing the SIOCSPARTNERINFO ioctl prior to the SIOCGPARTNERINFO ioctl can provide better performance, potentially eliminating wait time when issuing the SIOCGPARTNERINFO ioctl. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for details.

## Characteristics and restrictions

The argument is limited to 51 200 bytes.

## Examples

See [“BPX1IOC \(w\\_ioctl\) example” on page 1253](#) and [“BPX4IOC \(w\\_ioctl\) example” on page 1347](#).

# **\_\_wlm (BPX1WLM, BPX4WLM) – WLM interface service**

---

## Function

The \_\_wlm callable service invokes a wide variety of workload management (WLM) functions. You can also use it to invoke Enterprise Workload Manager (eWLM) ARM (Application Response Measurement) functions.

For information about the ARM functions, see *IBM Tivoli® eWorkload Management Version 1*.

## Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1WLM):</b>	31-bit
<b>AMODE (BPX4WLM):</b>	64-bit
<b>ASC mode:</b>	Primary mode



**Operation****Environment****Interrupt status:**

Enabled for interrupts

**Locks:**

Unlocked

**Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1WLM,(FunctionCode,
               ParmListPtr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4WLM with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

**Parameters****FunctionCode**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword that contains a value that indicates the type of WLM or eWLM function that the caller is requesting. The following are the supported values:

Value	Description
<b>WLM_QUERY_METRICS</b>	Query WLM System Information
<b>WLM_QUERY_SCHEDENV</b>	Query WLM Scheduling Environment
<b>WLM_CHECK_SCHEDENV</b>	Check WLM Scheduling Environment
<b>WLM_DISCONNECT</b>	Disconnect from WLM
<b>WLM_DELETE_WORKUNIT</b>	Delete a WLM Work Unit
<b>WLM_JOIN_WORKUNIT</b>	Join a WLM Work Unit
<b>WLM_LEAVE_WORKUNIT</b>	Leave a WLM Work Unit
<b>WLM_CONNECT_WORKMGR</b>	Connect to WLM as a work manager
<b>WLM_CONNECT_SERVERMGR</b>	Connect to WLM as a server manager
<b>WLM_CREATE_WORKUNIT</b>	Create a WLM work unit (this function creates an independent WLM enclave)
<b>WLM_CONTINUE_WORKUNIT</b>	Continue WLM work unit (this function creates a dependent WLM enclave)
<b>WLM_EXTRACT_WORKUNIT</b>	Extract the WLM work unit token (this function returns the WLM enclave token)
<b>WLM_EXPORT_WORKUNIT</b>	Export a WLM work unit
<b>WLM_UNDOEXPORT_WORKUNIT</b>	Undo a prior export request for a WLM work unit

Value	Description
<b>WLM_IMPORT_WORKUNIT</b>	Import a WLM work unit
<b>WLM_UNDOIMPORT_WORKUNIT</b>	Undo a prior import request for a WLM work unit
<b>WLM_QUERY_ENCLAVECLASS</b>	Query enclave class information for a WLM work unit
<b>WLM_CONNECT_EXPORTIMPORT</b>	Connect a subsystem to WLM to export and import work units, but not to create them
<b>ARM_BIND_THREAD</b>	Indicates that the calling thread is performing on behalf of an ARM transaction
<b>ARM_BLOCK_TRANSACTION</b>	Indicates that a started transaction is blocked waiting for an external transaction or some other event to complete
<b>ARM_DESTROY_APPLICATION</b>	Indicates that the registration data about an application is no longer needed
<b>ARM_DISCARD_TRANSACTION</b>	Signals that a started ARM transaction should be ignored
<b>ARM_GENERATE_CORRELATOR</b>	Generates an ARM correlator for use with ARM_REPORT_TRANSACTION
<b>ARM_GET_ARRIVAL_TIME</b>	Stores a 64-bit integer representing the current time
<b>ARM_REGISTER_APPLICATION</b>	Informs ARM of metadata about the application
<b>ARM_REGISTER_METRIC</b>	Informs ARM of metadata about each metric the application provides
<b>ARM_REGISTER_TRANSACTION</b>	Informs ARM of metadata about the transaction measured by the application
<b>ARM_REPORT_TRANSACTION</b>	Reports statistics about a transaction that has already completed
<b>ARM_START_APPLICATION</b>	Indicates that an instance of an application has started running and is prepared to make ARM calls
<b>ARM_START_TRANSACTION</b>	Indicates that a transaction is beginning execution
<b>ARM_STOP_APPLICATION</b>	Indicates that the application instance is finished making ARM calls
<b>ARM_STOP_TRANSACTION</b>	Signals the end of a transaction
<b>ARM_UNBIND_THREAD</b>	Indicates that the calling thread is no longer performing on behalf of an ARM transaction
<b>ARM_UNBLOCK_TRANSACTION</b>	Indicates that a transaction is no longer waiting for a downstream transaction to complete
<b>ARM_UPDATE_TRANSACTION</b>	Signals that a transaction is still processing
<b>EWLM_CLASSIFY_CORRELATOR</b>	Creates an eWLM specific ARM correlator for classification purpose

These constants are defined in the BPXYWLM macro; see [“BPXYWLM — WLM constants and parameter list DSECTs”](#) on page 1173.

For detailed information about the ARM function codes, see *IBM Tivoli eWorkload Management Version 1*.

### **ParmListPtr**

Supplied parameter

#### **Type:**

Address

#### **Length:**

Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the parameter list for the WLM function that is to be performed. See [“BPXYWLM — WLM constants and parameter list DSECTs”](#) on page 1173 for the mapping of the parameter lists for the various WLM functions.

### **Return\_value**

Returned parameter

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword in which the \_\_wlm service returns the return value for the WLM function that was requested.

For the following set of WLM functions, the service returns 0 if the request is successful, or -1 if it is not successful:

- WLM\_CHECK\_SCHEDENV
- WLM\_DISCONNECT
- WLM\_DELETE\_WORKUNIT
- WLM\_JOIN\_WORKUNIT
- WLM\_LEAVE\_WORKUNIT
- WLM\_CREATE\_WORKUNIT
- WLM\_CONTINUE\_WORKUNIT
- WLM\_QUERY\_METRICS
- WLM\_QUERY\_SCHEDENV
- WLM\_EXTRACT\_WORKUNIT
- WLM\_EXPORT\_WORKUNIT
- WLM\_UNDOEXPORT\_WORKUNIT
- WLM\_IMPORT\_WORKUNIT
- WLM\_UNDOIMPORT\_WORKUNIT
- WLM\_QUERY\_ENCLAVECLASS
- ARM\_BIND\_THREAD
- ARM\_BLOCK\_TRANSACTION
- ARM\_DESTROY\_APPLICATION
- ARM\_DISCARD\_TRANSACTION
- ARM\_GENERATE\_CORRELATOR
- ARM\_GET\_ARRIVAL\_TIME
- ARM\_REGISTER\_APPLICATION

- ARM\_REGISTER\_METRIC
- ARM\_REGISTER\_TRANSACTION
- ARM\_REPORT\_TRANSACTION
- ARM\_START\_APPLICATION
- ARM\_START\_TRANSACTION
- ARM\_STOP\_APPLICATION
- ARM\_STOP\_TRANSACTION
- ARM\_UNBIND\_THREAD
- ARM\_UNBLOCK\_TRANSACTION
- ARM\_UPDATE\_TRANSACTION
- EWLM\_CLASSIFY\_CORRELATOR

If the WLM\_QUERY\_METRICS, WLM\_QUERY\_SCHEDENV, or WLM\_QUERY\_ENCLAVECLASS function fails with an error that indicates that the supplied buffer was too small, the supplied length field in the input parameter list is updated to contain the length that is required for the function to succeed.

For the following set of WLM functions, the service returns a WLM connect token if the request is successful, or -1 if it is not successful:

- WLM\_CONNECT\_WORKMGR
- WLM\_CONNECT\_SERVERMGR
- WLM\_CONNECT\_EXPORTIMPORT

**Return\_Code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_wlm service stores the return code. The \_\_wlm service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The \_\_wlm service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EFAULT</b>	An argument of this service contained an address that was not accessible to the caller.
<b>EINVAL</b>	The FunctionCode parameter contains a value that is not correct; or the function parameter list data is not correct.
<b>EMVSWLMERROR</b>	A WLM service failed.  Consult Reason_code to determine the WLM service that failed and the reason for the error.
<b>EMVSARMERROR</b>	An ARM error occurred.  Consult Reason_code to determine the reason for the error. The ARM reason codes are documented in the _Elmarm4.h header file.
<b>EPERM</b>	The calling thread's address space is not permitted to the BPX.WLMSEVER FACILITY class profile. The caller's address space must be permitted to the BPX.WLMSEVER FACILITY class profile. If BPX.WLMSEVER is not defined, the calling process is not defined as a superuser (UID=0).

Return_code	Explanation
<b>EMVSSAF2ERR</b>	An error occurred in the security product.
<b>ESRCH</b>	A WLM_EXTRACT_WORKUNIT request was issued, but the WLM enclave token was not returned. Consult Reason_code to determine the exact reason it was not returned. Most likely, the unit of work is not in an enclave.
<b>EMVSERR</b>	Recovery processing was entered for a reason other than EFAULT.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the \_\_wlm service stores the reason code. The \_\_wlm service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For the WLM reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

ARM reason codes are documented in the \_Elmarm4.h header file.

**Usage notes**

1. The WLM\_CONNECT\_WORKMGR and WLM\_CONNECT\_EXPORTIMPORT functions both enable use of the export and import functions, but only the former enables use of the create function.
2. For a WLM\_CREATE\_WORKUNIT function invocation, some of the classification data that is pointed to by the supplied IWMCLSFY parameter list is truncated if it exceeds the maximum supported length, as follows:

Data	Maximum length
<b>ACCTINFO</b>	143 bytes
<b>SUBSYSPM</b>	255 bytes
<b>SOURCELU</b>	17 bytes
<b>COLLECTION</b>	18 bytes
<b>CORRELATION</b>	12 bytes

**Related services**

None.

**Characteristics and restrictions**

1. Certain \_\_wlm functions require that the caller have read access to the BPX.WLMSEVER FACILITY class profile, or a UID of 0 if the BPX.WLMSEVER FACILITY class profile is not defined. The following table shows the authorization required for each \_\_wlm function:

Table 34. Authorization requirements for __wlm functions	
Function	Authorization
<b>WLM_QUERY_METRICS</b>	No authorization required
<b>WLM_CONNECT_SERVERMGR</b>	
<b>WLM_EXTRACT WORKUNIT</b>	

Table 34. Authorization requirements for __wlm functions (continued)	
Function	Authorization
<b>WLM_CONNECT_EXPORTIMPORT</b>	If the caller has not already made a WLM_CONNECT_EXPORTIMPORT call, read access to the BPX.WLMSERVER FACILITY or UID 0 are not required
<b>WLM_DELETE_WORKUNIT</b>	If the caller has made a WLM_CONNECT_EXPORTIMPORT call, special authorization is not required.
<b>WLM_QUERY_SCHEDENV</b> <b>WLM_CHECK_SCHEDENV</b> <b>WLM_DISCONNECT</b> <b>WLM_JOIN_WORKUNIT</b> <b>WLM_LEAVE_WORKUNIT</b> <b>WLM_CONNECT_WORKMGR</b> <b>WLM_CREATE WORKUNIT</b> <b>WLM_IMPORT_WORKUNIT</b> <b>WLM_QUERY_ENCLAVECLASS</b> <b>WLM_UNDOIMPORT_WORKUNIT</b>	Read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.
<b>WLM_CONTINUE WORKUNIT</b>	A process can have one dependent enclave active at a time without authorization. If a process needs to have more than one dependent enclave active at the same time, it must have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.
<b>WLM_EXPORT_WORKUNIT</b>	A process can export the enclave it created using WLM_CONTINUE_WORKUNIT without authorization. To export some other enclave, the process must have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.
<b>WLM_UNDOEXPORT_WORKUNIT</b>	A process can undo its prior WLM_EXPORT_WORKUNIT request without authorization. To export some other enclave, the process must have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.

2. All ARM services, with the exception of ARM\_GET\_ARRIVAL\_TIME, require read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.

### Examples

See “BPX1WLM (\_\_WLM) example ” on page 1309.

# w\_piocctl (BPX1PIO, BPX4PIO) – Path name I/O control

## Function

The w\_piocctl callable service conveys a command to the physical file system that owns the specified file.

## Requirements

Operation	Environment
Authorization	Supervisor state or problem state, any PSW key
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE (BPX1PIO)	31-bit
AMODE (BPX4PIO)	64-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

## Format

```
CALL BPX1PIO, (Pathname_length,
               Pathname,
               Command,
               Argument_length,
               Argument,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4PIO with the same parameters.

## Parameters

### Pathname\_length

Supplied parameter.

#### Type

Integer

#### Length

Fullword

The name of a fullword that contains the length of the Pathname of the file.

### Pathname

Supplied parameter.

#### Type

Character string.

#### Character set

No restriction.

#### Length

Specified by the Pathname\_length parameter.

The name of a field that contains the name of the file to be acted upon.

**Command**

Supplied parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword that contains the command to be passed to the Physical File System.

**Argument\_length**

Parameter supplied and returned.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword containing the length of the argument. The length of the argument is specified as an integer value in the range 0-51 200.

**Argument**

Parameter supplied and returned.

**Type**

Defined by the Physical File System.

**Character set**

No restriction.

**Length**

Specified by the Argument\_length parameter.

Specifies the name of a buffer that contains the argument to be passed to the Physical File System.

For extended attributes, the area is mapped by the attx structure in the same BPXYATT mapping. Storage for the attribute value must be contiguous with the attx structure and the attributes\_length argument is the length of the attx structure plus the length of the attribute value.

**Return\_value**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the w\_piocctl service returns 0 if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the w\_piocctl service stores the return code. The w\_piocctl service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes* The w\_piocctl service can return one of the following values in the Return\_code parameter:



Return_code	Explanation
EACCES	The calling process does not have search permission for some component of the Pathname prefix or does not have permission to perform the requested function against the specified file.
EALREADY	An attempt was made to unregister a file that is not registered.
EINVAL	An incorrect Command or Argument_length was specified, or the function was directed against a character special file, or an invalid message queue was specified for the command of Ioctl#RegFileInt.  The following reason codes can accompany the return code: JRInvIoctlCmd, JRIOBufLengthInvalid, JRNotSupportedForFiletype, and JRipcBadId.
EINVAL	The extended attribute operation is invalid due to the values being passed. The input length for the ATTX parameter might be insufficient or the subcommand AttxSubCmd is invalid. The following reason codes can accompany the return code: JRInvMinArgLenAttx, JRInvArgLenAttx, JRInvAttxCmd.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
EMVSPARM	Incorrect parameters were passed to the service. The following reason codes can accompany the return code: JRNoStorage and JRInvParmLength.
ENAMETOOLONG	Pathname is longer than 1023 characters or some component of the pathname is longer than 255 characters. Name truncation is not supported.
ENOATTR	For the get function, the specified attribute is not set. For list, no attributes are set. The following reason codes can accompany the return code: JrAttxUnavail, JRAttxNoSup, and JrAttxName.
ENODEV	The device is incorrect. The function is not supported for this file. The following reason code can accompany the return code: JRFuncNotSupported.
ENOENT	No file named Pathname was found; or no path name was specified. The following reason code can accompany the return code: JrFileNotThere.
ENOTDIR	A component of the Pathname prefix is not a directory.
ERANGE	The buffer length that is provided in AttxValLn is not large enough to return the attribute or list. The following reason code can accompany the return code: JrInvAttxValLen.

**Reason\_code**

Returned parameter.

**Type**

Integer.

**Length**

Fullword.

The name of a fullword in which the w\_piocctl service stores the reason code. The w\_piocctl service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

## Extended attribute notes for w\_piocctl

1. The extended attributes functions are invoked by using the IOCTL command code for **extattr**, IOC#ATTR, which is in BPXYIOCC. Use the ATTX control block in the BPXYATT macro as the Argument parameter.
2. Only the Attx#scGet and Attx#scList functions are handled by this function. The ATTX mapping is variable length, with the **AttxVal** field being a char(\*) value.
  - For the get function, the caller fills in AttxName and AttxVal by specifying the attribute name and length. The AttxValLn field can be set to zero to get the size needed. It can also be set to the size of the space available to contain the attribute's value. On return, if AttxValLn was zero on input then Return\_value is set to the AttxValLn size needed to return the attribute. Otherwise, if the attribute's value fits within AttxValLn the value is returned starting at AttxVal and Return\_Value is set to the length of the value. If the attribute's value does not fit, then Return\_value is set to -1 and Return\_Code is set to ERANGE.
  - For the list function, the caller's values of AttxName and AttxNameLn are not used. The AttxVal field refers to the space to receive the list. The AttxValLn field can be set to zero in which case Return\_Value is set to the size required for the current list. If AttxValLn is set to nonzero, it should be set to the size of the area to contain the returned list of attribute names. On return, if the list fits within the provided space, the list is placed in AttxVal and Return\_Value is set to the length of the returned list. If the list does not fit, then Return\_Value is set to -1 and Return\_Code is set to ERANGE.
3. The get function extracts the attribute information that also exists in BPXYSTAT. The following table provides a correspondence between the information in BPXYATT and BPXYSTAT.

Table 35. Correspondence between BPXYATT and BPXYSTAT			
Xattr name	Existing SET attribute in BPXYATT	Existing GET attribute in BPXYSTAT	Restrictions and comments
trusted.apfauth	attapfauth	ST_APFAUTH	Set and remove functions that are limited according to "general flags" restrictions.
trusted.sharelib	attsharelib	ST_SHARELIB	Set and remove functions that are limited according to "general flags" restrictions.
trusted progctl	attprogctl	ST_PROGCTL	Set and remove functions that are limited according to "general flags" restrictions.
system.noshareas	attnoshareas	ST_NOSHAREAS	Set and remove functions that are limited according to "general flags" restrictions.
system.filefmt	attfilefmt	ST_FILEFMT	None.
system.filetag	attfiletag	ST_FILETAG	None.
system.seclabel	attseclabel	ST_SECLABEL	None.
system.useraudit	attuseraudit	ST_USERAUDIT	None.
system.auditoraudit	attauditoraudit	ST_AUDITORAUDIT	None.
system.auditid	Not an attribute. Equivalent to field ST_AUDITID in BPXYSTAT.	ST_AUDITID	Read-only.

Table 35. Correspondence between BPXYATT and BPXYSTAT (continued)

Xattr name	Existing SET attribute in BPXYATT	Existing GET attribute in BPXYSTAT	Restrictions and comments
system.dmodelacl	Not an attribute. Equivalent to field.	ST_DMODELACL	Read-only.
system.fmodelacl	Not an attribute. Equivalent to field.	ST_FMODELACL	Read-only.
system.accessacl	Not an attribute. Equivalent to field.	ST_ACCESSACL	Read-only.
system.createtime	Not an attribute. Equivalent to field.	ST_CREATETIME	Read-only.
user.*	No equivalent.	No equivalent.	Needs specific PFS support.

### Usage notes

1. This form of ioctl cannot be used with character special files. Refer to “w\_ioctl (BPX1IOC, BPX4IOC) — Control I/O ” on page 1003 for these files.
2. One of the uses of this function is to edit the access control lists of DFS remote files, and to register interest in files by path name.
3. This function can also be used to set or get the access options for z/OS UNIX files and directories. For more information, see the usage notes for “w\_ioctl (BPX1IOC, BPX4IOC) — Control I/O ” on page 1003 for descriptions of the SetfACL and GetfACL commands.
4. For the command of Ioccc#RegFileInt, see [usage note 7](#) in “w\_ioctl (BPX1IOC, BPX4IOC) — Control I/O ” on page 1003.

### Characteristics and restrictions

The argument is limited to 51 200 bytes.

## w\_statvfs (BPX1STF, BPX4STF) — Get the file system status

### Function

The w\_statvfs callable service obtains status information about a specified file system. You specify the file system by its file system name.

### Requirements

#### Operation

#### Authorization:

#### Dispatchable unit mode:

#### Cross memory mode:

#### AMODE BPX1STF):

#### AMODE BPX4STF):

#### ASC mode:

#### Interrupt status:

#### Locks:

#### Environment

Supervisor state or problem state, any PSW key

Task

PASN = HASN

31-bit

64-bit

Primary mode

Enabled for interrupts

Unlocked

**Operation****Environment****Control parameters:**

All parameters must be addressable by the caller and in the primary address space.

**Format**

The syntax format is as follows:

```
CALL BPX1STF, (File_system_name,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4STF with the same parameters.

**Parameters****File\_system\_name**

Supplied parameter

**Type:**

Character string

**Character set:**

Printable characters

**Length:**

44 bytes

The name of 44-character field that identifies the file system whose status is to be returned. The name must be left-justified and padded on the right with blanks.

This is the file system name as specified on the mount.

**Status\_area\_length**

Supplied parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword containing the length of the area to which the service returns status information.

**Status\_area**

Parameter supplied and returned

**Type:**

Structure

**Length:**

Specified by the Status\_area\_length parameter

The name of an area of length Status\_area\_length to which the service returns the status information for the file system. The BPXYSSTF macro maps this area. For information on this macro, see [“BPXYSSTF — Map response structure for file system status” on page 1161](#).

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the w\_statvfs service returns the length of the status written to the Status\_area if the request is successful, or -1 if it is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the w\_statvfs service stores the return code. The w\_statvfs service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The w\_statvfs service can return one of the following values in the Return\_code parameter:

Return_code	Explanation
<b>EAGAIN</b>	Information is temporarily unavailable. This can occur because the mount process for the file system is incomplete.
<b>EINVAL</b>	Parameter error; for example, File_system_name was not found. The following reason code can accompany the return code: JRFileSysNotThere.

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the w\_statvfs service stores the reason code. The w\_statvfs service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

1. It is not considered an error if the passed Status\_area\_length is not sufficient to hold all the returned information. (That is, future expansion is allowed for.) As much information as will fit is written to Status\_area, and this amount is returned.
2. If a buffer of length of zero is passed to this service, no data is returned and the return value is zero. You can check for the existence of a file system by passing such a length.
3. The amount of valid data returned in the Status\_area is indicated by the Return\_value. This allows for differences in the release levels of z/OS UNIX and the physical file systems.

**Related services**

- [“fstatvfs \(BPX1FTV, BPX4FTV\) — Get the file system status” on page 244](#)
- [“statvfs \(BPX1STV, BPX4STV\) — Get the file system status ” on page 896](#)

**Characteristics and restrictions**

None.

**Examples**

See [“BPX1STF \(w\\_statvfs\) example” on page 1297](#).

## write (BPX1WRT, BPX4WRT) – Write to a file or a socket

---

### Function

The write callable service writes data from a buffer to an open file or socket.

### Requirements

<b>Operation</b>	<b>Environment</b>
<b>Authorization</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode</b>	Task
	SRB - AF_INET/AF_INET6 socket support only
<b>Cross memory mode</b>	PASN = HASN
<b>AMODE (BPX1WRT)</b>	31-bit task or SRB mode
<b>AMODE (BPX4WRT)</b>	64-bit task or SRB mode
<b>ASC mode</b>	Primary mode
<b>Interrupt status</b>	Enabled for interrupts
<b>Locks</b>	Unlocked
<b>Control parameters</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1WRT, (File_descriptor,
               Buffer_address,
               Buffer_ALET,
               Write_count,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4WRT with the same parameters. The `Buffer_address` parameter is a doubleword.

### Parameters

#### File\_descriptor

Supplied parameter.

##### Type:

Integer

##### Length:

Fullword

The name of a fullword that contains the file descriptor of the file or socket to write to.

#### Buffer\_address

Supplied parameter.

##### Type:

Address

##### Length:

Fullword (doubleword)

The name of a fullword (doubleword) that contains the starting address of the data that is to be written.

### **Buffer\_ALET**

Supplied parameter.

#### **Type:**

Integer

#### **Length:**

Fullword

The name of a fullword that contains the ALET for Buffer\_address, which identifies the address space or data space the buffer resides in.

You should specify a Buffer\_ALET of 0 for the normal case of a buffer in the user's address space (current primary address space). If a value other than 0 is specified for the Buffer\_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

### **Write\_count**

Supplied parameter.

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword that contains the number of bytes that are to be written.

### **Return\_value**

Returned parameter.

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword in which the write service returns the number of actual bytes that were written, if the request is successful, or -1, if it is not successful.

### **Return\_code**

Returned parameter.

#### **Type**

Integer

#### **Length**

Fullword

The name of a fullword in which the write service stores the return code. The write service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The write service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EAGAIN</b>	Blocking is not in effect for the specified file, and output cannot be written immediately.
<b>EBADF</b>	The File_descriptor parameter does not contain the descriptor of an open file; or that file is not opened for write services. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	Connection reset by peer. The following reason code can accompany the return code: JRSocketNotCon.

Return_code	Explanation
<b>EFBIG</b>	Writing to the specified file would exceed either the file size limit for the process or the maximum file size that is supported by the physical file system.
<b>EINTR</b>	The service was interrupted by a signal before it could write any data. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	The Write_Count parameter contains a value that is less than zero.
<b>EIO</b>	The process is in a background process group and is attempting to write to its controlling terminal. However, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU signals, and the process group of the process is orphaned. This can happen, for example, if a background job tries to write to the terminal after the user has logged off.
<b>EMSGSIZE</b>	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutOfSocketCells.
<b>ENOTCONN</b>	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
<b>EPIPE</b>	The request is for a write to a pipe that is not open for reading by any other process; or an attempt was made to write to a socket that is shut down or closed. This error also generates a SIGPIPE signal. The following reason code can accompany the return code: JRSocketClosed.
<b>EWouldBlock</b>	<ul style="list-style-type: none"> <li>The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available.</li> <li>The socket is marked blocking. The call is blocked, without sending any data, for that time period that was specified in the SO_SNDTIMEO option.</li> </ul> <p>The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.</p>

**Reason\_code**

Returned parameter.

**Type**

Integer

**Length**

Fullword

The name of a fullword in which the write service stores the reason code. The write service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes in z/OS UNIX System Services Messages and Codes](#).

**Usage notes**

1. See [Chapter 3, “Callable services for SRB mode routines,” on page 1041](#) for more information about programming considerations for SRB mode.
2. The value of Write\_count is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.

The value of Write\_count is checked against the file size limit for the process. If no data can be written without exceeding this limit, an error of EFBIG is returned and the SIGXFSZ signal is generated



for the process. If at least one byte can be written before exceeding the file size limit, the write is considered successful.

3. If `File_descriptor` specifies a regular file or any other type of file on which you can seek, the write service begins writing at the file offset that is associated with that file descriptor. A successful write operation increments the file offset by the number of bytes that are written. If the incremented file offset is greater than the previous length of the file, the file is extended; the length of the file is set to the new file offset.

If the file descriptor refers to a file on which you cannot seek, the service begins writing at the current position. No file offset is associated with such a file.

If the file was opened with the "append" option, the write routine sets the file offset to the end of the file before it writes output.

4. Ordinarily, the number of bytes written to the output file is the number you specify in the `Write_count` parameter. (This number can be zero. If you ask to write zero bytes, the service simply returns a return value of zero without attempting any other action.)

If the write count that you specify is greater than the remaining space on the output device, or greater than the file size limit for the process, fewer bytes than you requested are written. When at least 1 byte is written, the write is considered successful. If you are not using a pseudoterminal, an attempt to append to the same file causes an error. An error of `ENOSPC` is returned when there is no remaining space on the output device. An error of `EFBIG` is returned when the file size limit for the physical file system is exceeded. An error of `EFBIG` is also returned if the file size limit for the process is exceeded, at which time the write service also generates a `SIGXFSZ` signal for the process. With a pseudoterminal, if there is not enough room in the buffer for the whole write, the number of bytes that fit are written, and the number of bytes written is returned. However, on the next write (assuming the buffer is still full), there is a block or `EAGAIN` is returned, depending on whether the file was opened blocking or nonblocking.

Similarly, fewer bytes are written if the service is interrupted by a signal after some, but not all, of the specified number of bytes are written. The return value shows the number of bytes that are written. But if no bytes were written before the routine was interrupted, the return value is -1, and an `EINTR` error is reported.

5. The write service causes signal `SIGTTOU` to be sent if all the following conditions are met:

- The process is attempting to write to its controlling terminal.
- `TOSTOP` is set as a terminal attribute (see [“tcgetattr \(BPX1TGA, BPX4TGA\) — Get the attributes for a terminal”](#) on page 926 or [“tcsetattr \(BPX1TSA, BPX4TSA\) — Set the attributes for a terminal”](#) on page 937).
- The process is running in a background process group.
- The `SIGTTOU` signal is not blocked or ignored.
- The process is not an orphan.

6. If automatic conversion is enabled with Unicode Services (ALL), see [Using Unicode Services](#) in *z/OS UNIX System Services Planning* for more information.

## Related services

- [“fcntl \(BPX1FCT, BPX4FCT\) — Control open file descriptors”](#) on page 216
- [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset”](#) on page 406
- [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511
- [“pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe”](#) on page 551
- [“read \(BPX1RED, BPX4RED\) — Read from a file or socket”](#) on page 652

**Note:** The write service is not related to the **write** shell command.

## Characteristics and restrictions

If the file was opened by an authorized program, all subsequent reads and writes against the file must be issued from an authorized state.

The read (BPX1RED, BPX4RED) and write (BPX1WRT, BPX4WRT) callable services do not support simultaneous reading or writing of the same shared open file by different threads when one or both of the following are true:

1. Automatic conversion is enabled using Enhanced ASCII (ON) and different character set IDs (CCSIDs) are used.
2. Automatic conversion is enabled using Unicode Services (ALL) and different CCSIDs are used, or mixing read and write operations of multibyte characters are performed which result is storing of partial characters.

The first restriction is not applicable if each thread coordinates its reads and writes so that simultaneous I/O does not occur. Both restrictions are not applicable if each thread opens the file independently.

Reads or writes that cause a conversion of greater than 2 G result in an EINVAL error with reason JzUniOpTooBig.

Refer to “[lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset](#)” on page 406 for instances of how a lseek operation in a conversion environment can affect read and write operations

## Examples

See “[BPX1WRT \(write\) example](#)” on page 1310.

## writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers

### Function

The writev callable service writes data from a set of buffers.

### Requirements

Operation	Environment
<b>Authorization:</b>	Supervisor state or problem state, any PSW key
<b>Dispatchable unit mode:</b>	Task
	SRB - AF_INET/AF_INET6 socket support only
<b>Cross memory mode:</b>	PASN = HASN
<b>AMODE (BPX1WRV):</b>	31-bit task or SRB mode
<b>AMODE (BPX4WRV):</b>	64-bit task mode only
<b>ASC mode:</b>	Primary mode
<b>Interrupt status:</b>	Enabled for interrupts
<b>Locks:</b>	Unlocked
<b>Control parameters:</b>	All parameters must be addressable by the caller and in the primary address space.

### Format

The syntax format is as follows:

```
CALL BPX1WRV, (File_descriptor,
               Iov_count,
```

```
Iov_struct,  
Iov_alet,  
Iov_buffer_alet,  
Return_value,  
Return_code,  
Reason_code)
```

AMODE 64 callers use BPX4WRV with the same parameters. All addresses in the Iov\_struct structure are doublewords.

## Parameters

### File\_descriptor

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a fullword that contains the file descriptor for which the writev is to be done.

### Iov\_count

Supplied and returned parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the number of buffers that are pointed to by Iov\_struct. The total number of buffers may not exceed IOV\_MAX (defined in [“BPXYIOV — Map the I/O vector structure”](#) on page 1098).

### Iov\_struct

Supplied parameter

#### Type:

Character

#### Length:

$\text{iov\_count} \times \text{length}(\text{iov})$

The name of a field that contains 31(64)-bit pointers to buffers from which data is to be retrieved for the purpose of writing to the file or socket. In 64-bit mode, Iov\_struct contains doubleword pointer and length subfields. See [“BPXYIOV — Map the I/O vector structure”](#) on page 1098 for more information about the format of this field.

### Iov\_alet

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the ALET for Iov\_struct.

### Iov\_buffer\_alet

Supplied parameter

#### Type:

Integer

#### Length:

Fullword

The name of a field that contains the ALET for buffers that are pointed to by Iov\_struct.

**Return\_value**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the writew service returns one of the following:

- The number of bytes that were written from the buffers, if the request is successful.
- -1, if the request is not successful.

**Return\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the writew service stores the return code. The writew service returns Return\_code only if Return\_value is -1. For a list of return code values, see [Return codes \(errno\)](#) in *z/OS UNIX System Services Messages and Codes*. The writew service can return one of the following values in the Return\_code parameter:

<b>Return_code</b>	<b>Explanation</b>
<b>EBADF</b>	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
<b>ECONNRESET</b>	Connection reset by peer. The following reason code can accompany the return code: JRSocketNotCon.
<b>EFBIG</b>	Writing to the specified file would exceed either the file size limit for the process, or the maximum file size supported by the physical file system.
<b>EINTR</b>	A signal interrupted the writew service before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
<b>EINVAL</b>	An incorrect value was specified on one of the input parameters. The following reason code can accompany the return code: JRSocketCallParmError.
<b>EIO</b>	The process is in a background process group and is attempting to write to its controlling terminal. However, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU signals, and the process group of the process is orphaned. This can happen, for example, if a background job tries to write to the terminal after the user has logged off.
<b>EMSGSIZE</b>	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
<b>ENOBUFS</b>	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
<b>ENOTCONN</b>	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
<b>ENOTSOCK</b>	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

Return_code	Explanation
<b>EPIPE</b>	An attempt was made to write to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed.  This error also generates a SIGPIPE signal.
<b>EPROTOTYPE</b>	An incorrect socket type was supplied. The following reason code can accompany the return code: JRIncorrectSocketType.
<b>EWouldBlock</b>	<ul style="list-style-type: none"> <li>The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available.</li> <li>The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option.</li> </ul> <p>The following reason codes can accompany the return code: JRTimeout, JRWouldBlock.</p>

**Reason\_code**

Returned parameter

**Type:**

Integer

**Length:**

Fullword

The name of a fullword in which the writew service stores the reason code. The writew service returns Reason\_code only if Return\_value is -1. Reason\_code further qualifies the Return\_code value. For a list of reason codes, see [Reason codes](#) in *z/OS UNIX System Services Messages and Codes*.

**Usage notes**

- See Chapter 3, “Callable services for SRB mode routines,” on page 1041 for more information about programming considerations for SRB mode.
- This callable service works with any open file descriptor, including files and sockets.
- Number of bytes written:** If the number of bytes to be written is greater than the remaining space on the output device, or greater than the file size limit for the process, not all of the data can be written. When at least 1 byte is written, the writew is considered successful. The return value shows the number of bytes that were written. An attempt to writew again to the same file causes an EFBIG error, and if the process file size limit has been exceeded, the writew service generates a SIGXFSZ signal for the process.
- Bytes written:** The number of bytes that are requested for writing is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.

The number of bytes that are requested for writing is checked against the file size limit for the process. If no data can be written without exceeding this limit, an error of EFBIG is returned and the SIGXFSZ signal is generated for the process. If at least one byte can be written before exceeding the file size limit, the write is considered successful.

- File offset:** If File\_descriptor specifies a regular file or any other type of file on which you can seek, the write service begins writing at the file offset that is associated with that file descriptor. A successful write operation increments the file offset by the number of bytes that are written. If the incremented file offset is greater than the previous length of the file, the file is extended; the length of the file is set to the new file offset.

If the file descriptor refers to a file on which you cannot seek, the service begins writing at the current position. No file offset is associated with such a file.

If the file was opened with the "append" option, the write routine sets the file offset to the end of the file before writing output.

6. **Number of bytes written:** Ordinarily, the number of bytes that are written to the output file is the number requested for writing. (This number can be zero. If you ask to write zero bytes, the service simply returns a return value of zero without attempting any other action.)

If the write count that you specify is greater than the remaining space on the output device, or greater than the file size limit for the process, fewer bytes than you requested are written. When at least 1 byte is written, the write is considered successful. If you are not using a pseudoterminal, an attempt to append to the same file causes an error. An error of ENOSPC is returned when there is no remaining space on the output device. An error of EFBIG is returned when the file size limit for the physical file system is exceeded. An error of EFBIG is also returned if the file size limit for the process is exceeded, at which time the write service also generates a SIGXFSZ signal for the process. With a pseudoterminal, if there is not enough room in the buffer for the whole write, the number of bytes that fit are written, and the number of bytes that were written is returned. However, on the next write (assuming the buffer is still full) there is a block or EAGAIN is returned, depending on whether the file was opened blocking or nonblocking.

Similarly, fewer bytes are written if the service is interrupted by a signal after some, but not all, of the specified number of bytes are written. The return value shows the number of bytes that were written. But if no bytes were written before the routine was interrupted, the return value is -1 and an EINTR error is reported.

7. The writew service causes SIGTTOU signal to be sent if all the following conditions are met:

- The process is attempting to write to its controlling terminal.
- TOSTOP is set as a terminal attribute (see [“tcgetattr \(BPX1TGA, BPX4TGA\) — Get the attributes for a terminal”](#) on page 926 or [“tcsetattr \(BPX1TSA, BPX4TSA\) — Set the attributes for a terminal”](#) on page 937).
- The process is running in a background process group.
- The SIGTTOU signal is not blocked or ignored.
- The process is not an orphan.

## Related services

- [“readv \(BPX1RDV, BPX4RDV\) — Read data and store it in a set of buffers”](#) on page 673
- [“write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket”](#) on page 1032

## Characteristics and restrictions

None.

## Examples

See [“BPX1WRV \(writew\) example”](#) on page 1310.

## Chapter 3. Callable services for SRB mode routines

### Overview of callable services available to SRB mode routines

A subset of the callable services is now available to SRB mode routines. Supported callable services can be called from SRBs using the same conventions that are used when calling them from task mode routines. However, unlike task mode routines, SRBs do not cause process dubbing on the first issue of a callable service. In order to issue callable services, they must be associated with a dubbed process; that is, the SRB must be running in a dubbed address space. Upon issuing the callable service, it must place into register 2 the address of the OAPB control block that represents the associated process. When the OAPB address in register 2 is zero, the SRB is associated with the initial process in the address space.

Most applications consist of a single process per address space. These applications should default to the initial or only process in the address space, and set register 2 to zero when invoking a callable service. Applications creating multiple processes per address space most likely need to explicitly provide the address of the OAPB of the process to which the SRB is to be associated. In this case, the SRB typically receives the OAPB address from the routine scheduling the SRB. The OAPB address is obtained from the PRLI control block, which contains process-related control information that is intended for external use. The PRLI is addressed as follows:

```
TcbStcb -> StcbOtcb -> OtcbOapb
```

The caller must be dubbed (the StcbOtcb address is nonzero).

The TCB referenced must represent a thread of the process to which the SRB is to be associated, and the PrliOapb field contains the address that must be passed by the SRB in register 2 when issuing a callable service. Note that the StcbOtcb field is zero until the task has been dubbed. The Otcb, Thli, and Prli are mapped by BPXZOTCB, BPXYTHLI, and BPXYPRLI, respectively.

A restriction on the use of callable services by an SRB is that the SRB must be running in non-cross memory mode (primary=secondary=home).

### Recovery

The use of callable services from SRB routines requires that the SRB and associated task mode routines must assume responsibility for certain recovery actions. Failure to provide for this can result in unwanted and unpredictable system problems; the system will take a dump. This responsibility revolves around the creation and termination of the process with which the SRB is associated. The process should be created (dubbed) prior to the scheduling of any SRBs that may be associated with it for the purpose of issuing calls. In addition, the SRBs must not be allowed to issue calls after the process has terminated, and the owner of the function taking advantage of SRB mode calls is responsible for guaranteeing that this does not happen. The function must also ensure that it cannot terminate until all of the application-created SRBs have completed processing.

Task and address space-level resource managers can be used to help meet this responsibility. You can use the MVS RESMGR service to set up task and address space resource managers. The following example shows the proper order of processing for the task mode routine, and for the task and address space dynamic resource manager.

### Task mode routine responsibilities

- Get the task dubbed by issuing a callable service. The task may already have been dubbed by having been pthread-created.
- Establish a task and an address space dynamic resource manager using the MVS RESMGR service. There are several RESMGR options you can choose when creating a resource manager. It is recommended that you choose to monitor only the address space containing the process, in order

to limit system overhead during termination of other address spaces; and that you monitor the top task of the process. Note that the resource manager must be established via the RESMGR service; and that this must be done after the task has been dubbed, or your resource manager will be called after the systems resource manager responsible for process-level termination.

- Schedule one or more SRBs, passing the OAPB address obtained from the PRLI.
- Ensure that all SRBs have completed, and that they will not issue any more callable services.
- Undub or terminate the task.

## Task and address space dynamic resource manager

- Terminate any SRBs that have not yet been dispatched via the MVS PURGEDQ service. You can provide filters to this service to purge SRBs selectively; for example, a multiprocess application could use the RMTR address filter to purge only SRBs for the terminating process.
- Wait for already-dispatched SRBs to complete.

For information about scheduling an SRB and SRB processing, see [Using a service request block \(SRB\) in z/OS MVS Programming: Authorized Assembler Services Guide](#).

## Callable services supported in SRB mode

---

The following callable services support SRB mode callers. The support of SRB mode callers was intended for the use of sockets from within SRB routines. Some of the following calls support files as well as sockets. These services will only support sockets from an SRB and not file operations. The callable services that are limited are so marked. The results of calling an unsupported callable service are unpredictable.

**Note:** AF\_UNIX sockets do not support SRB mode, only AF\_INET and AF\_INET6.

The following callable services are supported for 31-bit AMODE SRB callers:

- accept (BPX1ACP)
- accept\_and\_recv (BPX4ANR)
- asyncio (BPX1AIO)
- bind (BPX1BND)
- bind2addrsel (BPX1BAS)
- close (BPX1CLO) - socket support only
- connect (BPX1CON)
- gethostid & gethostname (BPX1HST)
- getpeername & getsockname (BPX1GNM)
- getsockopt & setsockopt (BPX1OPT)
- listen (BPX1LSN)
- msgsnd (BPX1QSN) - send to a message queue
- pfctl (BPX1PCT)
- read (BPX1RED) - socket support only
- readv (BPX1RDV) - socket support only
- recv (BPX1RCV)
- recvfrom (BPX1RFM)
- recvmsg (BPX1RMS)
- send (BPX1SND)
- sendmsg (BPX1SMS)
- sendto (BPX1STO)



- server\_init (BPX1SIN)
- setpeer (BPX1SPR)
- shutdown (BPX1SHT)
- socket & socket\_pair (BPX1SOC)
- srx\_np (BPX1SRX)
- w\_ioctl (BPX1IOC) - socket support only
- write (BPX1WRT) - socket support only
- writev (BPX1WRV) - socket support only

The following callable services are supported for 64-bit AMODE SRB callers. The support is intended for the use of sockets from within SRB routines. Some of the following calls support files as well as sockets. These services only support sockets from an SRB and do not support file operations. The callable services that are limited are so marked. The results of calling an unsupported callable service are unpredictable.

- asyncio (BPX4AIO)
- accept\_and\_recv (BPX1ANR)
- read (BPX4RED) - socket support only
- recv (BPX4RCV)
- recvmsg (BPX4RMS)
- sendmsg (BPX4SMS)
- write (BPX4WRT) - socket support only



## Appendix A. System control offsets to callable services

An alternative to loading or link-editing the service stub is to include in the code the system control offset to the callable service. For example, use decimal 52 for the offset of access (BPX1ACC).

When using the offsets, set up the registers as follows:

### Register 1

Contains the address of your parameter list. Set bit 0 of the last address in the list on.

### Register 14

Contains the return address in the invoking module.

### Register 15

Contains the address of the callable service code.

## Example

The following is an example of code that specifies the offset. The example assumes that register 1 is set up with the address of the parameter list. Replace *offset* with the appropriate value from the following offset table.

```
L    15,16      CVT - common vector table
L    15,544(15) CSRTABLE
L    15,24(15)  CSR slot
L    15,offset(15) Address of the service
BALR 14,15      Branch and link
```

## List of offsets

Table 36. System control offsets to callable services

Service	Offset	Function
BPX1AC4	1228	accept4
BPX1ACC	52	access
BPX1ACK	972	auth_check_resource_np
BPX1ACP	508	accept
BPX1AIO	988	asyncio
BPX1ALR	224	alarm
BPX1ANR	1060	accept_and_recv
BPX1ASP	1088	aio_suspend
BPX1ATM	668	attach_execmvs
BPX1ATX	664	attach_exec
BPX1AUT	1328	__authenticate
BPX1BAS	592	bind2addrsel
BPX1BND	512	bind
BPX1CCA	480	cond_cancel
BPX1CCS	1012	console_np

Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1CHA	84	chaudit
BPX1CHD	56	chdir
BPX1CHM	60	chmod
BPX1CHO	64	chown
BPX1CHP	764	chpriority
BPX1CHR	500	chattr
BPX1CID	968	convert_id_np
BPX1CLD	68	closedir
BPX1CLN	1292	clone
BPX1CLO	72	close
BPX1CMA	1284	fchmodat
BPX1COA	1268	fchownat
BPX1CON	516	connect
BPX1CPL	1132	__cpl
BPX1CPO	484	cond_post
BPX1CRA	1316	chatrat
BPX1CRT	872	chroot
BPX1CSE	488	cond_setup
BPX1CTW	492	cond_timed_wait
BPX1CWA	496	cond_wait
BPX1DEL	888	deletehfs
BPX1DSD	1124	sw_sigdlv
BPX1EFD	1220	eventfd
BPX1ENV	960	oe_env_np
BPX1EPC	1196	epoll_create
BPX1EPL	1200	epoll_ctl
BPX1EPW	1204	epoll_wait
BPX1EXC	228	exec
BPX1EXI	232	_exit
BPX1EXM	236	execmvs
BPX1EXT	200	extlink_np
BPX1FAA	1280	faccessat
BPX1FAI	1168	freeaddrinfo
BPX1FCA	140	fchaudit
BPX1FCD	852	fchdir
BPX1FCM	88	fchmod

Table 36. System control offsets to callable services (continued)

<b>Service</b>	<b>Offset</b>	<b>Function</b>
BPX1FCO	92	fchown
BPX1FCR	504	fchattr
BPX1FCT	96	fcntl
BPX1FLK	1240	flock
BPX1FPC	100	fpathconf
BPX1FRK	240	fork
BPX1FSA	1320	fstatat
BPX1FST	104	fstat
BPX1FSY	108	fsync
BPX1FTR	112	ftruncate
BPX1FTV	848	fstatvfs
BPX1GAI	1164	getaddrinfo
BPX1GCL	1024	getclientid
BPX1GCW	116	getcwd
BPX1GEG	244	getegid
BPX1GEP	860	getpgid
BPX1GES	864	getsid
BPX1GET	736	w_getipc
BPX1GEU	248	geteuid
BPX1GGE	772	getgrent
BPX1GGI	252	getgrgid
BPX1GGN	256	getgrnam
BPX1GGR	260	getgroups
BPX1GHA	1160	gethostbyaddr
BPX1GHN	1156	gethostbyname
BPX1GID	264	getgid
BPX1GIV	1028	givesocket
BPX1GLG	268	getlogin
BPX1GMN	76	w_getmntent
BPX1GNI	1172	getnameinfo
BPX1GNM	524	getpeername
BPX1GPE	776	getpwent
BPX1GPG	272	getpgrp
BPX1GPI	276	getpid
BPX1GPN	280	getpwnam
BPX1GPP	284	getppid

Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1GPS	428	w_getpsent
BPX1GPT	916	grantpt
BPX1GPU	288	getpwuid
BPX1GPY	744	getpriority
BPX1GRD	1308	getrandom
BPX1GRL	820	getrlimit
BPX1GRU	824	getrusage
BPX1GTH	1056	__getthent
BPX1GTR	752	getitimer
BPX1GUG	292	getugrps
BPX1GUI	296	getuid
BPX1GWD	936	getwd
BPX1HST	520	gethostid
BPX1IAW	1212	inotify_add_watch
BPX1INI	1208	inotify_init
BPX1IOC	120	w_ioctl
BPX1IPT	396	mvsiptaffinity
BPX1IRW	1216	inotify_rm_watch
BPX1ITY	12	isatty
BPX1KIL	308	kill
BPX1LCO	832	lchown
BPX1LCR	1180	lchattr
BPX1LDX	1192	loadhfs_ext
BPX1LNA	1272	linkat
BPX1LNK	124	link
BPX1LOD	880	loadhfs
BPX1LSK	128	lseek
BPX1LSN	532	listen
BPX1LST	132	lstat
BPX1MAT	720	shmat
BPX1MCT	724	shmctl
BPX1MDA	1252	mkdirat
BPX1MDT	728	shmdt
BPX1MGT	732	shmget
BPX1MKD	136	mkdir
BPX1MKN	144	mknod

Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1MMI	1136	__map_init
BPX1MMP	796	mmap
BPX1MMS	1140	__map_service
BPX1MNA	1256	mknodat
BPX1MNT	148	mount
BPX1MP	688	mvspause
BPX1MPC	408	mvspocclp
BPX1MPI	680	mvspauseinit
BPX1MPR	800	mprotect
BPX1MSD	336	mvsunsigsetup
BPX1MSS	312	mvssigsetup
BPX1MSY	804	msync
BPX1MUN	808	munmap
BPX1NIC	748	nice
BPX1OAT	1244	openat
BPX1OPD	152	opendir
BPX1OPN	156	open
BPX1OPT	528	getsockopt
BPX1OSE	1100	__osenv
BPX1PAF	1072	__pid_affinity
BPX1PAS	316	pause
BPX1PCF	160	pathconf
BPX1PCT	768	pfsctl
BPX1PI2	1236	pipe2
BPX1PIO	984	w_pioclt
BPX1PIP	164	pipe
BPX1POE	1176	__poe
BPX1POL	932	poll
BPX1PQG	1152	pthread_quiesce_and_get_np
BPX1PRC	1224	prctl
BPX1PRL	1288	prlimit
BPX1PSI	460	pthread_setintr
BPX1PST	472	pthread_setintrtype
BPX1PTB	448	pthread_cancel
BPX1PTC	432	pthread_create
BPX1PTD	444	pthread_detach

Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1PTI	476	pthread_testintr
BPX1PTJ	440	pthread_join
BPX1PTK	464	pthread_kill
BPX1PTQ	412	pthread_quiesce
BPX1PTR	320	ptrace
BPX1PTS	452	pthread_self
BPX1PTT	1016	pthread_tag_np
BPX1PTX	436	pthread_exit_and_get
BPX1PVR	1248	pivot_root
BPX1PWD	788	password
BPX1QCT	692	msgctl
BPX1QDB	948	querydub
BPX1QGT	696	msgget
BPX1QRC	700	msgrcv
BPX1QSE	388	quiesce
BPX1QSN	704	msgsnd
BPX1RCV	540	recv
BPX1RD2	856	readdir2
BPX1RDD	168	readdir
BPX1RDL	172	readlink
BPX1RDV	536	readv
BPX1RDX	940	read_extlink
BPX1RED	176	read
BPX1REN	180	rename
BPX1RFM	544	recvfrom
BPX1RLA	1260	readlinkat
BPX1RMD	188	rmdir
BPX1RMG	8	resource
BPX1RMS	548	recvmsg
BPX1RNA	1276	renameat2
BPX1RPH	884	realpath
BPX1RW	1108	pread
BPX1RWD	184	rewinddir
BPX1SA2	1084	__sigactionset
BPX1SCT	708	semctl
BPX1SDD	300	setdubdefault



Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1SEC	1044	__security
BPX1SEG	424	setegid
BPX1SEL	552	select
BPX1SEU	420	seteuid
BPX1SF	1064	send_file
BPX1SGE	780	setgrent
BPX1SGI	328	setgid
BPX1SGQ	1104	sigqueue
BPX1SGR	792	setgroups
BPX1SGT	712	semget
BPX1SHN	1304	sethostname
BPX1SHT	572	shutdown
BPX1SIA	324	sigaction
BPX1SIN	1004	server_init
BPX1SIP	340	sigpending
BPX1SLK	1068	__shm_lock
BPX1SLP	344	sleep
BPX1SMC	1112	__smc
BPX1SMF	1036	__smf_record
BPX1SMS	560	sendmsg
BPX1SND	556	send
BPX1SNS	1296	setns
BPX1SOC	576	socket_pair
BPX1SOP	716	semop
BPX1SPB	416	sigputback
BPX1SPE	784	setpwent
BPX1SPG	348	setpgid
BPX1SPM	352	sigprocmask
BPX1SPN	760	spawn
BPX1SPR	568	setpeer
BPX1SPW	1008	server_pwu
BPX1SPY	740	setpriority
BPX1SRG	896	setregid
BPX1SRL	816	setrlimit
BPX1SRU	892	setreuid
BPX1SRX	1080	srx_np

Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1SSI	356	setsid
BPX1SSU	360	sigsuspend
BPX1STA	192	stat
BPX1STE	1076	set_timer_event
BPX1STF	80	w_statvfs
BPX1STL	684	set_thread_limits
BPX1STO	564	sendto
BPX1STQ	1144	server_threads_query
BPX1STR	756	setitimer
BPX1STV	844	statvfs
BPX1STW	1096	sigtimedwait
BPX1SUI	364	setuid
BPX1SWT	468	sigwait
BPX1SYA	1264	symlinkat
BPX1SYC	368	sysconf
BPX1SYF	1312	syncfs
BPX1SYM	196	symlink
BPX1SYN	868	sync
BPX1TAF	1148	mvstthreadaffinity
BPX1TAK	1032	takesocket
BPX1TDR	24	tcdrain
BPX1TFH	20	tcflush
BPX1TFW	28	tcflow
BPX1TGA	32	tcgetattr
BPX1TGC	900	tcgetcp
BPX1TGP	36	tcgetpgrp
BPX1TGS	912	tcgetsid
BPX1TIM	372	times
BPX1TLS	964	pthread_security_np
BPX1TRU	828	truncate
BPX1TSA	40	tcsetattr
BPX1TSB	44	tcsendbreak
BPX1TSC	904	tcsetcp
BPX1TSP	48	tcsetpgrp
BPX1TST	908	tcsettables
BPX1TYN	16	ttyname

Table 36. System control offsets to callable services (continued)

Service	Offset	Function
BPX1ULA	1324	unlinkat
BPX1UMK	204	umask
BPX1UMT	208	umount
BPX1UNA	376	uname
BPX1UNL	212	unlink
BPX1UNS	1300	unshare
BPX1UPT	920	unlockpt
BPX1UQS	392	unquiesce
BPX1UTI	216	utime
BPX1VAC	944	v_access
BPX1VCL	1188	v_close
BPX1VCR	620	v_create
BPX1VEX	876	v_export
BPX1VGA	632	v_getattr
BPX1VGT	596	v_get
BPX1VIO	400	v_ioctl
BPX1VLK	604	v_lookup
BPX1VLN	640	v_link
BPX1VLO	660	v_lockctl
BPX1VMD	624	v_mkdir
BPX1VOP	1184	v_open
BPX1VPC	1040	v_pathconf
BPX1VRA	616	v_readlink
BPX1VRD	612	v_readdir
BPX1VRE	644	v_rmdir
BPX1VRG	584	v_reg
BPX1VRL	600	v_rel
BPX1VRM	648	v_remove
BPX1VRN	652	v_rename
BPX1VRP	588	v_rpn
BPX1VRW	608	v_rdwr
BPX1VSA	636	v_setattr
BPX1VSF	656	v_fstatfs
BPX1VSY	628	v_symlink
BPX1WAT	380	wait
BPX1WLM	1048	__wlm

*Table 36. System control offsets to callable services (continued)*

<b>Service</b>	<b>Offset</b>	<b>Function</b>
BPX1WRT	220	write
BPX1WRV	580	writew
BPX1WTE	840	wait-extension
BPX2ITY	928	isatty2
BPX2MNT	1128	__mount
BPX2OPN	1052	openstat
BPX2RMS	976	recvmsg2
BPX2SMS	980	sendmsg2
BPX2TYN	924	ttyname2

## Appendix B. Mapping macros—AMODE 31

Mapping macros map the parameter options in many callable services. The fields with the comment "Reserved for IBM Use" are not programming interfaces. A complete list of the options for each macro is listed in the macro in [“Macros mapping parameter options” on page 1055](#).

Most of the mapping macros can be expanded with or without a DSECT statement. The invocation operand DSECT=YES (default) can be used with either reentrant or nonreentrant programs with the appropriate rules governing the storage backed by the USING statement.

Many of the mapping macros exploit the fact that DC expands as a DS in a DSECT and as a DC with its initialized value in a CSECT. When these fields are expanded as or within DSECT statement, the program is responsible for initializing the necessary fields.

### Macros mapping parameter options

Specifying DSECT=YES (the default for all macros) creates a DSECT. Addressability requires a USING and a register pointing to storage.

Specifying DSECT=NO (exceptions are listed when this is not allowed) allocates space in the current DSECT or CSECT. In reentrant programs, programmers can place these macros in the DSECT with DSECT=NO, and addressability is accomplished without the individual USING required by DSECT=YES. Nonreentrant programs can place their macros in the program's CSECT and addressability is obtained through the program base register(s).

Specifying LIST=YES (the default for most macros) causes the expansion of the macro to appear in the listing. You can override this by using PRINT OFF.

Specifying LIST=NO removes the macro expansion from the listing.

Additional keywords VARLEN and PREFIX are described in the individual sections where they apply.

### BPXYACC — Map flag values for access

```

                BPXYACC
** BPXYACC:  Access intent flags
** Used by: ACC
ACC             DSECT ,
ACCRSRV        DS    CL2    Reserved
ACCFLAGS       DS    XL1    Flags
ACCEFFID       EQU    X'04'  check effective ids
ACCDEVNO       EQU    X'02'  return devno if exists
ACCWAIT        EQU    X'01'  Wait for Async Mount
ACCINTENTFLAGS DS    XL1    Access Intent Flags
*             EQU    X'F0'    Reserved
ACC_F_OK       EQU    X'08'  Check for file existence
ACC_R_OK       EQU    X'04'  Check for read access to file
ACC_W_OK       EQU    X'02'  Check for write access to file
ACC_X_OK       EQU    X'01'  Check for execute access to file
ACC#LENGTH     EQU    *-ACC  Length of this structure
** BPXYACC End

```

### BPXYAIO — Map asyncio parameter list

AMODE 64 callers use [“BPXYAIO — Map asyncio parameter list” on page 1187](#).

```

                BPXYAIO ,
* ----- 31-Bit Version
* -----
** BPXYAIO:  Asyncio parameter block

```

```

** Used by: AIO
AIO          DSECT ,
AIOFD        DS      F      File Descriptor
AIOBUFFDW    DS      0CL8   Eight byte addresses
AIOBUFFALET  DS      F      Alet for AioBuffPtr
&AIOBUFFPTR31 DS      F      Buffer Pointer
AIOBUFFSIZE  DS      F      Buffer Length or Iov count
AIOOFFSETDW  DS      0CL8   Offset in File
AIOOFFSETH   DS      F      Offset in File highword
AIOOFFSET    DS      F      Offset in File lowword
AIOMSGEVENT  DS      0C     Message Event overlays SigEv
&AIOSIGEVENT31 DS      CL20  POSIX Signals
AIOREQPRI0   DS      F      REQUEST PRIORITY
AIOLIOOPCODE DS      F      LIO_LISTIO() OP
*
          ORG      AIOLIOOPCODE
AIOCMD       DS      F      Command Code
AIONOTIFYTYPE DS      H      Notification Type
AIOCFLAGS    DS      XL1    Control Flags
AIOOK2COMPIMD EQU     X'80'  Ok to complete immediately
AIOCALLB4    EQU     X'40'  Call exit before redrive
AIOSYNC      EQU     X'10'  Do synchronously
AIOEXITMODETCB EQU     X'08'  0=SRB, 1=TCB
AIOCANCELNOWAIT EQU     X'04'  Nowait option on cancel
AIOCANCELNONOTIFY EQU     X'02'  NoNotify option on cancel
AIOTCBAFFINITY EQU     X'01'  TCB Affinity I/O
AIOCFLAGS2   DS      XL1    Control Flags2
AIOUSERKEY   EQU     X'F0'  Caller's User's Key bit positions
AIOUSEUSERKEY EQU     X'08'  Use User's Key for moves
AIOTHLICOMECB EQU     X'04'  AioEcbPtr points tp ThliComEcb
AIOCOMMBUFF  EQU     X'02'  Common Area Buffer
AIOMSGIOVALET DS      F      Alet for recvmg/sendmsg IOV
AIOIOVBUFFALET DS      F      Alet for all IOV buffers
*
AIORV        DS      F      Return value
AIORC        DS      F      Return code
AIORSN       DS      F      Reason code
*
AIOPOSIXFLAGS DS      XL4    Posix flags
&AIOEXITPTR31 DS      F      Pointer to user exit
AIOEXITDATA   DS      CL8    User Data for exit program
AIOECBPTR     DS      F      ECB address
AIOSOCKADDRLEN DS      F      Sockaddr length
&AIOSOCKADDRPTR31 DS      F      Sockaddr pointer
AIOTIMEOUT    DS      F      Timeout Value in Milli-seconds
AIOACEE       DS      F      SRB ACEE for MLS
AIOSICODE     DS      XL2    Signal si_code
AIORES06      DS      CL2    Reserved
AIOLEN        DS      F      (Output,debug) Len of AIO rcvd
AIOENDVER1    DS      0D     End of Original Aiocb
* ----- 64-Bit Extension
AIOLP64       DS      0D
&AIOBUFFPTR64 DS      AD     Buffer Ptr
&AIOEXITPTR64 DS      AD     Exit Program Address
&AIOSIGEVENT64 DS      CL32   SigEvent Structure
&AIOSOCKADDRPTR64 DS      AD     Sockaddr Ptr
          DS      CL8
* ----- Version 3 Extension
* ----- 31-Bit Version
AIORES01      DS      F      RESERVED
AIOLOCSOCKADDRPTR DS      F      Local Sockaddr Ptr for ANR
AIOLOCSOCKADDRLEN DS      F      Local Sockaddr Len for ANR
AIOANR_SOCKET DS      F      Accepted Socket for ANR
          DS      CL48
AIOENDVER3    DS      0D     End of Version 3 extension
AIOEND        DS      0D     End of Aiocb
*
AIO#LENGTH    EQU     *-AIO  Length of this structure
*
** AIO command values
AIO#ACCEPT    EQU     126
AIO#CONNECT   EQU     128
AIO#READ      EQU     43
AIO#WRITE     EQU     54
AIO#READV     EQU     133
AIO#WRITEV    EQU     144
AIO#RECV      EQU     134
AIO#SEND      EQU     138
AIO#RECVFROM  EQU     135
AIO#SENDTO    EQU     140
AIO#RECVMSG   EQU     243

```

```

AIO#SENDMSG      EQU    244
AIO#ANR          EQU    264
AIO#BRLOCK       EQU    264
AIO#SELPOLL      EQU     2
AIO#CANCEL       EQU     1
*
** AIO notify type
AIO#POSIX        EQU     0
AIO#MVS          EQU     1
AIO#MSGQ         EQU     2
*
** AIO Message Event Structure
*      For AioNotifyType of AIO#MSGQ the AioMsgEvent
*      structure overlays AioSigEvent (31-bit location).
*      Msgbuf and Msgbuf64 are defined in BPXYMSG.
*      IPC_NOWAIT is defined in BPXYIPCP.
*
AIO_BEFORE_MSGEV DS     0C          Note current position
                ORG     AIOMSGEVENT
AIOMSGEV_QID     DS     F          Msg Queue Id
AIOMSGEV_SIZE    DS     H          Length of Msg_mtext
AIOMSGEV_FLAG    DS     H          0 or IPC_NOWAIT
AIOMSGEV_ADDR64  DS     D          Amode(64)-> MsgBuf64
                ORG     AIOMSGEV_ADDR64
AIOMSGEV_ADDRH   DS     F
AIOMSGEV_ADDR    DS     F          Amode(31)-> MsgBuf
                ORG     AIO_BEFORE_MSGEV      Return to above
*
AIO#MSGTEXTMAX   EQU    240          Max Msg_MText
*
** AIO Signal Event
SIGEVENT         DSECT ,
SIGEVENT         DS     0F
SIGEV_NOTIFY     DS     F          NOTIFICATION TYPE
SIGEV_SIGNO      DS     F          SIGNAL NUMBER
SIGEV_VALUE      DS     &AIOPTRSIZE SIG VALUE
                ORG     SIGEV_VALUE
SIVAL_INT        DS     F
                ORG     SIGEV_VALUE
SIVAL_PTR        DS     &AIOPTRSIZE
SIGEV_NOTIFY_FUNCTION DS &AIOPTRSIZE NOTIF. FUNCTION
SIGEV_NOTIFY_ATTRIBUTES DS &AIOPTRSIZE NOTIF. ATTRIBUTES
*
SIGEV#LENGTH     EQU    *-SIGEVENT  Length of this structure
*
*      SIGEV_NOTIFY Values
SIGEV_SIGNAL      EQU     0          GENERATE A SIGNAL
SIGEV_NONE        EQU     1          DON'T GENERATE SIGNAL
SIGEV_THREAD      EQU     2          Call Notif. function
*
** AIOTIMEOUT VALUES
AIO#FOREVER       EQU     0          NO TIMEOUT, JUST WAIT
AIO#NOWAITING     EQU    X'FFFFFFFF' NO WAITING, JUST CHECK
** AIO CANCEL RETURN VALUES
AIO_CANCELED      EQU     1          ALL CANCELS SUCCESSFUL
AIO_NOTCANCELED   EQU     2          AT LEAST 1 CANCEL FAILED
AIO_ALLDONE       EQU     3          NONE CANCELED, ALL COMP
*
** BPXYAIO End

```

## BPXYATT — Map file attributes for chatr and fchatr

```

MACRO
BPXYATT &DSECT=YES,&LIST=YES,&ATTX=NO,          x
        &ATTXValLength=0,&ATT=YES
GBLB   &ATT411
AIF    (&ATT411 EQ 1).E411
&ATT411 SETB 1
AIF    ('&LIST' EQ 'YES').A411
PUSH   PRINT BPXYATT: File attributes for chatr system call
PRINT  OFF
AGO    .A411
*
.A411  ANOP
AIF    ('&ATT' EQ 'NO').J411
** BPXYATT: File attributes for chatr system call
** Used By: CHR FCR
AIF    ('&DSECT' EQ 'NO').B411

```

```

ATT          AGO      .C411      DSECT ,
.B411        ANOP      ,

ATT          DS      0D
          DC      XL(ATT#LENGTH)'00'
          ORG      ATT

.C411        ANOP      ,
ATTBEGIN      DS      0D
*
ATTID          DC      C'ATT ' Eye Catcher
ATTVERSION      DC      AL2(ATT#VER)
*
ATTRES01      DS      CL2      Reserved
ATTSETFLAGS      DS      0XL4      Flags - which fields to set
ATTSETFLAGS1      DS      X      Flag byte 1
ATTMODECHG      EQU      X'80'      1 = Change to the mode indicated
ATTOWNERCHG      EQU      X'40'      1 = Change to Owner indicated
ATTSETGEN      EQU      X'20'      1 = Set General attributes
ATTTRUNC      EQU      X'10'      1 = Truncate size
ATTATIMECHG      EQU      X'08'      1 = Change the Atime
ATTATIMETOD      EQU      X'04'      1 = Change to the Current Time
ATTMTIMECHG      EQU      X'02'      1 = Change the Mtime
ATTMTIMETOD      EQU      X'01'      1 = Change to the Current Time
ATTSETFLAGS2      DS      X      Flag byte 2
ATTMAAUDIT      EQU      X'80'      1 = Modify auditor audit info
ATTMUAUDIT      EQU      X'40'      1 = Modify user audit info
ATTCTIMECHG      EQU      X'20'      1 = Change the Ctime
ATTCTIMETOD      EQU      X'10'      1 = Change Ctime to the Current
*
ATTREFTIMECHG      EQU      X'08'      1 = Change the RefTime
ATTREFTIMETOD      EQU      X'04'      1 = Change RefTime to Current Time
ATTFILEMTCHG      EQU      X'02'      1 = Change File Format
ATTRES04      EQU      X'01'      Reserved
ATTSETFLAGS3      DS      X      Reserved
ATTRES05      EQU      X'80'      Reserved
ATTCHARSETIDCHG      EQU      X'40'      1 = Change File tag
ATTLP64TIMES      EQU      X'20'      1 = Use 64-bit times
ATTSECLABELCHG      EQU      X'10'      1 = Set Seclabel
ATTSETFLAGS4      DS      X      Reserved
ATTMODE      DS      F      File Mode, mapped by BPXYMODE
ATTUID      DS      F      User ID of the owner of the file
ATTGID      DS      F      Group ID of the Group of the file
ATTGENMASK      DS      0XL4      Mask to indicate which General
*
*
*
ATTOPAQUEMASK      DS      XL3      Opaque attribute flags - Reserved
*
*
ATTVISIBLEMASK      DS      X      Visible attribute flags
ATTNODEFILESMASK      EQU      X'20'      Files should not be deleted
ATTSHARELIBMASK      EQU      X'10'      Shared Library
ATTNOSHAREMASK      EQU      X'08'      No shareas flag
ATTAPFAUTHMASK      EQU      X'04'      APF authorized flag
ATTPROGCTLMASK      EQU      X'02'      Program controlled flag
ATTGENVALUE      DS      0XL4      General attribute values
*
*
ATTOPAQUE      DS      XL3      Opaque attribute flags - Reserved
*
*
ATTVISIBLE      DS      X      Visible attribute flags
ATTNODEFILES      EQU      X'20'      Files should not be deleted
ATTSHARELIB      EQU      X'10'      Shared Library
ATTNOSHAREAS      EQU      X'08'      No shareas flag
ATTAPFAUTH      EQU      X'04'      APF authorized flag
ATTPROGCTL      EQU      X'02'      Program controlled flag
ATTSIZE      DS      0D      File Size in bytes, for regular
*
*
ATTSIZE_H      DS      F      First word of size
ATTSIZE_L      DS      F      Second word of size
ATTATIME      DS      F      Time of last access
ATTMTIME      DS      F      Time of last data modification
ATTAUDITORAUDIT      DS      F      Area for auditor audit info
ATTUSERAUDIT      DS      F      Area for user audit info
ATTCTIME      DS      F      Time of last file status change
*
*
*
ATTREFTIME      DS      F      Reference time
ATTENDVER1      DS      0D      End of Version 1
ATTFILEFMT      DS      XL1      File Format
ATTRES02      DS      XL3      Reserved for future
ATTFILETAG      DS      F      File tag (see BPXYSTAT)
ATTRES03      DS      CL8      Reserved for future
ATTENDVER2      DS      0D      End of Version 2
*

```



```

ATTTIME64      DS    D    Access Time
ATTMTIME64     DS    D    Data Mod Time
ATTCTIME64     DS    D    Metadata Change Time
ATTREFTIME64   DS    D    Reference Time
ATTSECLABEL    DS    CL8   Security Label
ATTVER3RES02   DS    CL8   Reserved for R6
ATTENDVER3     DS    0D    End of Version 3
*
*   Constants
*
ATT#VER        EQU    ATT#VER03 Current version
ATT#VER01      EQU    1      Version 1 of this structure
ATT#VER02      EQU    2      Version 2 of this structure
ATT#VER03      EQU    3      Version 3 of this structure
ATT#LENGTH     EQU    *-ATTBEGIN                                     X
                                     Length of ATT
ATT#VER1LEN    EQU    ATTENDVER1-ATTBEGIN                             X
                                     Length of Version 1 ATT
ATT#VER2LEN    EQU    ATTENDVER2-ATTBEGIN                             X
                                     Length of Version 2 ATT
ATT#VER3LEN    EQU    ATTENDVER3-ATTBEGIN                             X
                                     Length of Version 3 ATT
.J411      AIF    ('&ATTX' NE 'YES').H411
*****
* ATTX Parameter List for Extended Attributes
*****
      AIF    ('&DSECT' EQ 'NO').F411
ATTX      DSECT ,
      AGO    .G411
.F411      ANOP ,
      DS    0D
ATTX      DC    XL(ATTXVAL-ATTXBEGIN-256)'00'
      DC    XL(256)'00'
      ORG    ATTX
      .G411      ANOP ,
ATTXBEGIN      DS    0D
*
ATTXID          DC    C'ATTX'      Eye Catcher
ATTXVERSION     DC    AL2(ATTX#VER)
ATTXSUBCMD      DS    H          Subcommand
ATTX#SCSET      EQU    1          Set
ATTX#SCGET      EQU    2          Get
ATTX#SCREMOVE   EQU    3          Remove
ATTX#SCLIST     EQU    4          List
ATTXFLAGS       DS    XL4        Flags
ATTXFLSET       EQU    X'00000004'
ATTXFREPLACE    EQU    X'00000002'
ATTXFCREATE     EQU    X'00000001'
      DS    XL60      Reserved
ATTXNAMELN      DS    F          Attribute name length
ATTXNAME        DS    CL256     Attribute name
ATTXVALLN       DS    F          Attribute value length
      AIF    ('&ATTXValLength' NE '0').I411
ATTXVAL         DS    0D        Attribute value
      AGO    .I411X
.I411      ANOP
ATTXVAL         DS    XL(&ATTXValLength)
.I411X      ANOP
*
*   Constants
*
ATTX#VER        EQU    1          Current version
ATTX#LENGTH     EQU    *-ATTXBEGIN
ATTX#MAXVAL     EQU    50000      Max Value Length
      AIF    ('&DSECT' NE 'NO').H411
AttX#ID         DC    C'ATTX'      Control Block id
* AttXName values
AttX#apfauth    DC    C'trusted.apfauth'
AttX#sharelib   DC    C'trusted.sharelib'
AttX#progctl    DC    C'trusted.progctl'
AttX#noshareas  DC    C'system.noshareas'
AttX#filefmt    DC    C'system.filefmt'
AttX#filetag    DC    C'system.filetag'
AttX#seclabel   DC    C'system.seclabel'
AttX#auditoraudit DC    C'system.auditoraudit'
AttX#useraudit  DC    C'system.useraudit'
AttX#accessacl  DC    C'system.accessacl'
AttX#fmodelacl  DC    C'system.fmodelacl'
AttX#dmodelacl  DC    C'system.dmodelacl'
AttX#createtime DC    C'system.createtime'
AttX#auditid    DC    C'system.auditid'
* class prefixes

```

```

Attx#trustedclass DC C'trusted.'
Attx#systemclass DC C'system.'
Attx#userclass DC C'user.'
.H411 ANOP ,
** BPXYATT End
SPACE 3
AIF ('&LIST' EQ 'YES').E411
POP PRINT
.E411 ANOP ,
MEND ,

```

## BPXYAUDT — Map flag values for chaudit and fchaudit

```

BPXYAUDT ,
** BPXYAUDT: External audit flags
** Used By: CHA, FCA
AUDT DSECT ,
AUDTREADACCESS DS XL1 Read Access Auditing Flags
AUDTREADFAIL EQU X'02' 1 = audit failing read accesses
AUDTREADSUCC EQU X'01' 1 = audit successful read accesses
AUDTWRITEACCESS DS XL1 Write Access Auditing Flags
AUDTWRITEFAIL EQU X'02' 1 = audit failing write accesses
AUDTWritesucc EQU X'01' 1 = audit successful write accesses
AUDTEXECACCESS DS XL1 Execute/Search Auditing Flags
AUDTEXECFAIL EQU X'02' 1 = audit failing exec or search
AUDTEXECsucc EQU X'01' 1 = audit successful exec or search
AUDTRSRV DS XL1 Flag byte 4 -Reserved
AUDT#LENGTH EQU *-AUDT Length of this structure
** BPXYAUDT End

```

## BPXYBRLK — Map byte range lock request for fcntl

```

BPXYBRLK ,
** BPXYBRLK: External Byte Range Locking interface control block
** Used By: FCT
BRLK DSECT ,
L_TYPE DS H Requested lock type:
F_RDLCK EQU 1 Shared or read lock
F_WRLCK EQU 2 Exclusive or write lock
F_UNLCK EQU 3 Unlock
L_WHENCE DS H Flag for starting offset
L_START DS 0CL8 Relative offset in bytes
L_START_H DS F High word of relative offset
L_START_L DS F Low word of relative offset
L_LEN DS 0CL8 Size of lock in bytes
L_LEN_H DS F High word of size of lock in bytes
L_LEN_L DS F Low word of size of lock in bytes
L_PID DS F Process ID of process holding lock
BRLK#LENGTH EQU *-BRLK Length of this area
** BPXYBRLK End

```

## BPXYCCA — Map input/output structure for \_\_console()

AMODE 64 callers use “BPXYCCA — Map input/output structure for \_\_console()” on page 1189.

```

BPXYCCA ,
** BPXYCCA: Msg Attributes for console_np service
** Used By: CCS
CCA DSECT ,
CCABEGIN DS 0D
*
CCAVERSION DC AL2(CCA#VER)
* Version of this structure

```

CCARES01	DS	CL2	Reserved	
CCAMSGLENGTH	DS	F	Length of msg pointed to by CCAMSGPTR	
CCAMSGPTR	DS	A	Pointer to Msg text	
CCARES02	DS	CL8	Reserved	
CCAENDVER1	DS	0F	End of Version 1	
CCASTARTVER2	DS	0F	Start of Version 2	
CCARES03	DS	F	Reserved	
CCAWTOPARMS	DS	0F	Start of WTO message attributes	
CCAROUTCDELIST	DS	A	Pointer to list of message routing codes	X
CCARES04	DS	F	Reserved	
CCADESCLIST	DS	A	Pointer to list of message descriptor codes	X
CCARES05	DS	F	Reserved	
CCAWMCSFLAGS	DS	0F	WTO MCS Flags	
CCAMCSFLAGB1	DS	XL1	MCS flags byte 1	
CCAHRDCPY	EQU	X'80'	Send message to hard copy log only	
CCAMCSFLAGB2	DS	XL1	MCS flags byte 2	
CCAMCSFLAGB3	DS	XL1	MCS flags byte 3	
CCAMCSFLAGB4	DS	XL1	MCS flags byte 4	
CCAWTOTOKEN	DS	F	Token for message to be issued	
CCAMSGIDPTR	DS	A	Pointer to location where message is stored by BPX1CCS	X
CCARES06	DS	F	Reserved	
CCARES07	DS	F	Reserved	
CCADOMPARMS	DS	0F	Delete message parameters	
CCADOMTOKEN	DS	F	Token of message(s) to be deleted	
CCAMSGIDLIST	DS	A	Pointer to list of message ids to be deleted	X
CCARES08	DS	F	Reserved	
CCAENDVER2	DS	0D	End of version 2	
CCASTARTVER3	DS	0CL40	Start of version 3	
CCAMODCARTPTR	DS	A	Pointer to 8 byte CART returned for MODIFY/STOP command	X
CCARES09	DS	CL4	Not used for amode 31	
CCAMODCONSOLEIDPTR	DS	A	Pointer to 4 byte ConsoleID returned for MODIFY/STOP command	X
CCARES10	DS	CL4	Not used for amode 31	
CCAMSGCART	DS	CL8	Supplied - CART to be specified on WTO when message is issued	X
CCAMSGCONSOLEID	DS	CL4	Supplied - ConsoleID to be specified on WTO when message is issued	X
CCARES11	DS	CL12	Reserved	
CCAENDVER3	DS	0D	End of version 3	
* Constants				
* CCA#VER EQU CCA#VER02 Current version				
CCA#VER01	EQU	1	Version 1 of this structure	
CCA#VER02	EQU	2	Version 2 of this structure	
CCA#VER03	EQU	3	Version 3 of this structure	
CCA#LENGTH	EQU	*-CCABEGIN	Length of CCA	X
CCA#VER1LEN	EQU	CCAENDVER1-CCABEGIN	Length of Version 1 CCA	X
CCA#VER2LEN	EQU	CCAENDVER2-CCABEGIN	Length of Version 2 CCA	X
CCA#VER3LEN	EQU	CCAENDVER3-CCABEGIN	Length of Version 3 CCA	X
** BPXYCCA End				

## BPXYCID – Map the returning structure for getclientid()

```

BPXYCID ,
*
*****
** BPXYCID: z/OS UNIX ClientId Structure *
** Used By: Sockets LFS *
*****
*
CID DSECT , ClientId structure
CIDBEGIN DS 0D
*
CIDDOMAIN DS F Domain
CIDNAME DS CL8 Address space name
CIDTASK DS CL8 Subtask name
CIDRESERVED DS CL20 Reserved

```

```

*
CID#LENGTH      EQU  *-CID      Constant - Fixed length of CID
*
CIDNAMEUPPER     ORG  CIDNAME
CIDPID           DS    F          Binary zeroes
*               DS    F          Process Id
*
CIDTYPE          ORG  CIDRESERVED
CIDSPECIFIC      DS    X          Type of request
*               DS    CL19
*
CIDSOCKTOKEN     ORG  CIDSPECIFIC
                  DS    CL3
                  DS    F          Returned token
*               ORG  ,
*
CID#CLOSE        EQU  1          Close socket
CID#SELECT       EQU  2          Giver will do select
*
*
*
** BPXYCID End

```

## BPXYCLNP— Map clone syscall parameters

```

.L0004  ANOP
CLNP_ID  DS    CL4      +00 'CLNP' block identifier
CLNP_LEN DS    H        +04 CLNP block length
CLNP_VER DS    H        +06 CLNP block version
CLNP_FLAGS DS BL4      +08 Input Flags. Flags are mapped by
                        Clone_Flags type in PL/X or CLONE_xxxxxx
                        constants in assembler
CLNP_EXIT_SIGNAL DS F   +0C Signal to deliver to the parent upon
                        termination of the child.
                        DS    CL16      +10 Reserved for future use
CLNP_ENDVER1 DS 0C      +20
*
CLNP#ID EQU  C'CLNP'   CLNP identifier
CLNP#LEN EQU  32        Length of CLNP
CLNP#VER EQU  1         Current version
CLNP#VER1 EQU  1        Version 1
CLNP#VER1LEN EQU 32     Length of Version 1
*
CLONE_NEWPID EQU
536870912

CLONE_NEWIPC EQU
134217728

CLONE_PARENT EQU 32768

```

## BPXYCONS — Constants used by services

BPXYCONS is composed only of EQUates. DSECT= is allowed but ignored.

```

** BPXYCONS: Syscall constants
** Used By: Many syscalls
DFLT_ARG_MAX     EQU  1048576  Constant for default ARG_MAX   @EGC
DFLT_CHILD_MAX   EQU    6      Constant for default CHILD_MAX
*               (_POSIX_CHILD_MAX)
DFLT_CLK_TCK     EQU   100     Constant for default CLK_TCK
*               (100 ticks per second)
DFLT_NGROUPS_MAX EQU   8191    Constant for default NGROUPS_MAX
*               (RACF Maximum value)
DFLT_OPEN_MAX    EQU    16     Constant for default OPEN_MAX
*               (_POSIX_OPEN_MAX)
DFLT_TZNAME_MAX  EQU    9      Constant for default TZNAME_MAX
DFLT_JOB_CONTROL EQU    1      Constant for default JOB_CONTROL
DFLT_SAVED_IDS   EQU    1      Constant for default SAVED_IDS
DFLT_VERSION     EQU  199009   Constant for default VERSION
DFLT_THREAD_TASKS_MAX_NP EQU 50 Constant default THREAD_TASKS_MAX_NP
DFLT_USERIDLEN_MAX EQU    8     Max characters for a userid   @DKA

```

```

DFLT_PASSWDLEN_MAX EQU 8      Max characters for a password      @DKA
DFLT_PASSWDPHRLEN_MAX EQU 100  Max characters for password phrase @EBA
DFLT_2_CHAR_TERM EQU 1      Constant default SC_2_CHAR_TERM @P1A
SPACE ,
* items from sysconf()
SC_ARG_MAX EQU 1      Constant for querying ARG_MAX
SC_CHILD_MAX EQU 2      Constant for querying CHILD_MAX
SC_CLK_TCK EQU 3      Constant for querying CLK_TCK
SC_JOB_CONTROL EQU 4      Constant for querying JOB_CONTROL
SC_NGROUPS_MAX EQU 5      Constant for querying NGROUPS_MAX
SC_OPEN_MAX EQU 6      Constant for querying OPEN_MAX
SC_SAVED_IDS EQU 7      Constant for querying SAVED_IDS
SC_TZNAME_MAX EQU 9      Constant for querying TZNAME_MAX
SC_VERSION EQU 10      Constant for querying VERSION
SC_THREAD_TASKS_MAX_NP EQU 11  Constant to query THREAD_TASKS_MAX_NP
SC_2_CHAR_TERM EQU 12      Constant for querying VERSION @P1A
SC_THREADS_MAX_NP EQU 13      Constant to query THREADS_MAX_NP @D5A
SC_MMAP_MEM_MAX_NP EQU 14      Constant to query MMAP_MEM_MAX_NP @DAA
SC_TTY_GROUP EQU 15      Constant to query TTY_GROUP @PEA
SC_PAGESIZE EQU 16      Constant to query Page Size @D2A
SC_PAGE_SIZE EQU 16      Constant to query Page Size @D2A
SPACE ,
* wait function code @DCA
#WAIT3 EQU 1      wait3() function code @DCA
#WAITID EQU 2      waitid() function code @DCA
#WAIT4 EQU 3      wait4() function code @FPA
SPACE ,
* items from waitf()
WNOHANG EQU 1      Wait, do not suspend execution
WUNTRACED EQU 2      Wait, return status of stopped child
WCONTINUED EQU 4      Wait, return status of continued child
* @DCA
WEXITED EQU 8      Wait for process that have exited
* @DCA
WSTOPPED EQU 16      Wait, return status of stopped child
* @DCA
WNOWAIT EQU 32      Wait, return status of a child without
* changing the state. The child can be
* waited for again. @DCA
SPACE ,
* waitid() id type options @DCA
P_PID EQU 0      Wait for the child with a process ID @DCA
* @DCA
P_PGID EQU 1      Wait for any child with a process
* group ID @DCA
P_ALL EQU 2      Wait for any child @DCA
SPACE ,
* BPX1PTX Options
PTEXITTHREAD EQU 0      Pthread exit
PTGETNEWTTHREAD EQU 1      Pthread get new
PTFAILIFLASTTHREAD EQU 2      Pthread fail if last thread @D4A
SPACE ,
QUIESCE_TERM EQU 1      Quiesce threads type = term @D3A
QUIESCE_FORCE EQU 2      Quiesce threads type = force @D3A
QUIESCE_QUERY EQU 3      Alias of pthread_query @P4C
PTHREAD_QUERY EQU 3      Quiesce threads type = query @P4A
QUIESCE_FREEZE EQU 4      Quiesce threads type = freeze @D6A
QUIESCE_UNFREEZE EQU 5      Quiesce threads type = unfreeze @D6A
FREEZE_THIS_THREAD EQU 6      Quiesce threads type = freezeme @D6A
* Skip 7 because of collision with BPXZCONS Freeze_Force
FREEZE_EXIT EQU 8      Quiesce threads type = freeze_exit
QUIESCE_SRB EQU 9      Quiesce threads type = SRBs @DGA
* Skip 10 and 11 due to collision with BPXZCONS Freeze/Unfreeze Fast
* @P6A
SPACE ,
PTHREAD_INTR_ENABLE# EQU 0      Cancel request type = enabled
PTHREAD_INTR_DISABLE# EQU 1      Cancel request type = disabled
PTHREAD_INTR_CONTROLLED# EQU 0      Cancel request type = controlled
PTHREAD_INTR_ASYNCHRONOUS# EQU 1      Cancel request type = Asynchronous
SPACE ,
STDIN_FILENO EQU 0      Standard input value, file descriptor
STDOUT_FILENO EQU 1      Standard output value, file descriptor
STDERR_FILENO EQU 2      Standard error value, file descriptor
SPACE ,
DUBTHREAD EQU 0      Dub a thread default setting @L1A
DUBPROCESS EQU 1      Dub a process default setting @L1A
DUBTASKACEE EQU 2      Dub a task ACEE setting @02A
DUBPROCESSDEFER EQU 4      Dub a process - but defer dub @04A
DUBNOSIGNALS EQU 8      Dub a process - no signals @DWA
DUBJOBPERM EQU 16      Dub as a permanent Job @DYA
DUBNOJSTUNDUB EQU 32      Dub process such that jobstep does not
* get undubbed when last non-jst undubs

```

```

*                                     @DYA
DUBABENDCALLS      EQU    64    Dub process such that system calls
*                                     abend during a shutdown/restart window
*                                     @DYA
DUBUNIQUEACEE      EQU    128   @07A
DUBFAILNOTREADY    EQU    256   Fail dub if kernel is not up    @F1A
*                                     SPACE ,
STL_MAX_TASKS      EQU    1     Replace MaxThreadTask only    @D7A
STL_MAX_THREADS    EQU    2     Replace MaxThreads only        @D7A
STL_SET_BOTH       EQU    3     Replace both limits            @D7A
*                                     SPACE ,
NICE_ZERO          EQU    20    Default Process Scheduling Priority
*                                     SPACE ,
PRIO_PROCESS       EQU    1     Looking for a specific process ID
PRIO_PGRP          EQU    2     Looking for processes in a process grp
PRIO_USER          EQU    3     Looking for processes for a user ID
*                                     SPACE ,
CPRIO_ABSOLUTE     EQU    1     Priority value is an absolute value
CPRIO_RELATIVE     EQU    2     Priority value is a relative value
*                                     SPACE ,
*   Define equates for memory map
PROT_READ          EQU    1     Mapped data can be read         @DAA
PROT_WRITE         EQU    2     Mapped data can be written      @DAA
PROT_NONE          EQU    4     Mapped data cannot be accessed  @DAA
PROT_EXEC          EQU    8     Mapped data can be executed (treated
*                                     as PROT_READ)              @DAA
*                                     SPACE ,
MAP_PRIVATE        EQU    1     Changes to the mapped data are private
MAP_SHARED         EQU    2     Changes to the mapped data are shared
MAP_FIXED          EQU    4     Interpret map address exactly   @DAA
MAP_MEGA           EQU    8     Use megabyte allocations        @D4A
MAP_64             EQU    16    Above the bar features          @FHA
MAP_ANONYMOUS      EQU    32    Map not backed by a file         @FSA
*                                     SPACE ,
MS_SYNC            EQU    1     Performs synchronous writes     @DAA
MS_ASYNC           EQU    2     Performs asynchronous writes    @DAA
MS_INVALIDATE      EQU    4     Invalidate the cached memory mapped
*                                     pages                        @DAA
*                                     SPACE ,
*   Define equates for spawn
SPAWN_FDCLOSED     EQU    -1    Do not inherit this file desc   @D9A
*                                     SPACE ,
RLIMIT_CORE        EQU    4     Limit size of core dump         @DBA
RLIMIT_CPU         EQU    0     Limit CPU time per process      @DBA
RLIMIT_FSIZE       EQU    1     Limit file size                 @DBA
RLIMIT_NOFILE      EQU    6     Limit number of open files      @DBA
RLIMIT_AS          EQU    5     Limit address space size        @DBA
RLIMIT_MEMLIMIT    EQU    7     Limit storage above the bar     @EOA
*                                     SPACE ,
RLIM_INFINITY      EQU    2147483647 No limit value            @DBA
*                                     SPACE ,
RUSAGE_SELF        EQU    0     Rusage for current process      @DBA
RUSAGE_CHILDREN    EQU    -1    Rusage for terminated children   @DBA
RUSAGE_OTHER       EQU    1     Rusage for another process      @FQA
*                                     SPACE ,
*   Define equates for querydub output status
QDB_DUBBED_FIRST   EQU    1     Task has already been dubbed.
*                                     This task and this RB caused the
*                                     dub.                        @DCA
QDB_DUBBED         EQU    2     Task has already been dubbed.
*                                     Other task or other RB caused
*                                     the dub                    @DCA
QDB_DUB_MAY_FAIL    EQU    4     Task has not been dubbed, but may
*                                     fail if attempted. Most likely
*                                     reason for failure will be a missing
*                                     or incomplete user security profile,
*                                     or OMVS segment not defined @DCA
QDB_DUB_OKAY       EQU    8     Task has not been dubbed, and should
*                                     succeed if attempted        @DCA
QDB_DUB_AS_PROCESS EQU    16    Task has not been dubbed, but its
*                                     address space has. New task will dub
*                                     as another process within the address
*                                     space                      @DCA
QDB_DUB_AS_THREAD   EQU    32   Task has not been dubbed, but its
*                                     address space has. New task will dub as
*                                     a thread within the process @DCA
*                                     SPACE ,
*   Define equates for oe_env_np syscall function codes
ENQWAIT_PROCESS    EQU    1     Examine/Change ENQ wait interruption
*                                     state                        @P6A
FREEZE_EXIT_REG    EQU    2     Register/deregister an exit
*                                     for pthread_quiesce(freeze_exit) @P6A

```

```

MVS_USERID      EQU 3    Retrieve MVS userid of invoker    @P7A
ENV_TOGGLE_SEC  EQU 4    Toggle btw task/process security @P7A
DFP_CLEANUP_EXIT_REG EQU 5 Register DFP Close cleanup exit @Q1A
BPXK_PARAMETER  EQU 6    Env Vars to Kernel                @P9A
ENV_STOR_SERVICE EQU 7    Swappable or Non-Swappable
*                                     address space          @DJA
QUICK_FREEZE_EXIT_REG EQU 8 Register/deregister an exit for
*                                     fast pthread_quiesce_and_get @DUA
SHUTDOWN_REG    EQU 9    Register/block for shutdown
*                                     processing              @DXA
WRITE_DOWN      EQU 10   Query/Alter write_down state of an
*                                     ACEE (MLS support)       @E5A
PIDXFER_QUERY   EQU 11   Query if process image is a result of
*                                     a PIDXFER               @DXA
QUERY_MODE      EQU 12   Query AMODE/RMODE/AMODE capability of
*                                     target proess           @E1A
MUST_STAY_CLEAN EQU 13   ENABLE/QUERY Must Stay Clean state of
*                                     the invoking process     @E9A
XPLATFORM_PROCESS EQU 14  ENABLE/QUERY XPlatform level for
*                                     the caller's process     @FRA
*
*       SPACE ,
*       Define equates for XPLATFORM_PROCESS function of BPX1ENV @FRA
*
XPLATFORM_MAX    EQU 1    Maximum supported XPLATFORM level
*                                     @FRA
*
*       SPACE ,
*       Define equates for possible options of ENV_STOR_SERVICE
*       For future additions, make equates multiples of 2
*
BPX_SWAP         EQU 1    Make the address space swappable    4@PED
BPX_NONSWAP      EQU 2    Make the address space non-swappable @PEC
*
*       SPACE ,
*
*       Define equates for possible options of MUST_STAY_CLEAN @E9A
* @E9A
MSC_QUERY        EQU 0    Query current Must Stay Clean state @E9A
MSC_ENABLE       EQU 1    Enable Must Stay Clean state        @E9A
*
*       Define possible output for MSC_QUERY option            @E9A
* @E9A
MSC_DISABLED     EQU 0    Query result: disabled              @E9A
MSC_ENABLED      EQU 1    Query result: enabled even accoss Job
*                                     Step termination          @E9A
MSC_ENABLED_COND EQU 2    Query result: enabled conditionally,
*                                     Job Step termination will disable @E9A
*
*       Define equates for all possible options for WRITE_DOWN function
*       code of BPX1ENV
*
WD_OQUERY        EQU 0    Query write_down                    @E5A
WD_ACTIVATE      EQU 1    Activate write_down                 @E5A
WD_INACTIVATE    EQU 2    Inactivate write_down               @E5A
WD_RESET         EQU 3    Reset write_down to default         @E5A
*
WD_SCOPE_AS      EQU 1    Target ACEE is AS                    @E5A
WD_SCOPE_THD     EQU 2    Target ACEE is task                  @E5A
*
WD_IS_ACTIVE      EQU 1    Query result: active                @E5A
WD_IS_INACTIVE    EQU 0    Query result: inactive              @E5A
*
*       Define equates for QUERY_MODE return values            @E1A
*
BIT24_MODE       EQU 1    24 bit AMODE, RMODE or AMODE cap. @E1A
BIT31_MODE       EQU 2    31 bit AMODE, RMODE or AMODE cap. @E1A
BIT64_MODE       EQU 3    64 bit AMODE, RMODE or AMODE cap. @E1A
AMODE_INITIALIZING EQU 4    AMODE for process not known yet @PNA
*
*       SPACE ,
*       Define equates for possible options of ENV_SHUTDOWN_REG
*
ENV_REGISTERBLOCK EQU 1    Register to Block Shutdown        @DYA
ENV_REGISTERPERMP EQU 2    Register as a Permanent job/proc   @DYA
ENV_DEREGISTERBLOCK EQU 3  Dereg as a blocking job/proc       @DYA
ENV_DEREGISTERPERM EQU 4   Dereg as a permanent job/proc      @DYA
ENV_REGISTERNOTIFY EQU 5    Register as a notify job/proc      @DYA
ENV_DEREGISTERNOTIFY EQU 6  Dereg as a notify job/proc        @DYA
ENV_REGISTERJOB    EQU 1    Register Job                       @DYA
ENV_REGISTERPROC   EQU 2    Register Process                   @DYA
*
*       SPACE ,
*       Define equates for PIDXFER_QUERY return values         @DXA
*

```

```

PIDXFER_YES      EQU 1    Process was PIDXFERed      @DXA
PIDXFER_NO       EQU 2    Process was not PIDXFERed   @DXC
SPACE ,
* Define equates for versions of OSMF on BPXESMF syscall
OSMF_VER_HOM1110 EQU 1    Version 1 of OSMF, for HOM1110 @P5A
OSMF_VER_HOM1120 EQU 2    Version 2 of OSMF, for HOM1120 @P5A
OSMF_VER_HOM1130 EQU 3    Version 3 of OSMF, for HOM1130 @P5A
SPACE ,
* Define equates for task security syscall function codes
TLS_CREATE_THREAD_SEC# EQU 1    Build Task Security      @P7A
TLS_DELETE_THREAD_SEC# EQU 2    Delete Task Security      @P7A
TLS_TASK_ACEE#        EQU 3    set posix identity from task ACEE@DYA
TLS_TASK_ACEE_USP#    EQU 4    User passed ACEE/USP pair   @E6A
TLS_DAEMON_THREAD_SEC# EQU 5    Build unauthenticated Security @E8A
TLS_IDENTITY_USERID#   EQU 1    User identity: 1-8 char userid @P7A
TLS_IDENTITY_UID#     EQU 2    User identity: 4-byte uid   @P7A
TLS_IDENTITY_CERT#    EQU 4    User identity: CERT structure @PFA
SPACE ,
* Define equates for __Security syscall
* __Security function code
SECURITY_CREATE#      EQU 1    Create new security environment @DKA
SECURITY_CERTREG#     EQU 2    Register certificate with caller @DOA
SECURITY_CERTDEREG#   EQU 3    DeReg certificate from caller @DOA
SECURITY_CERTAUTH#    EQU 4    Authorize certificate from caller@E3A
* __Security user identity
SECURITY_USERID#      EQU 1    User identity is 1-8 char userid @DKA
SECURITY_CERTIFICATE# EQU 2    User identity is a certificate @DKA
SPACE ,
* Define equates for convert_id_np (BPX1CID) syscall function codes
CID_GET_UUID#        EQU 1    Retrieve UUID              @P8A
CID_GET_USERID#      EQU 2    Retrieve userid             @P8A
SPACE ,
* Define equates for _pid_affinity (BPX1PAF) syscall function codes
PAF_ADD_PID#         EQU 1    Add PID to affinity list     @DMA
PAF_DELETE_PID#      EQU 2    Delete PID from affinity list @DMA
SPACE ,
* Define equates for auth_check_resource_np syscall access types
ACK_READ#           EQU 1    Test READ access             @P8A
ACK_UPDATE#         EQU 2    Test UPDATE access           @P8A
ACK_CONTROL#        EQU 3    Test CONTROL access          @P8A
ACK_ALTER#          EQU 4    Test ALTER access            @P8A
SPACE ,
* The high order two bytes of the reason codes returned by
* OpenMVS services contains a value that is used to qualify
* the contents of the low order two bytes. If the contents of
* the high-order two bytes are within the range of #CMID_LO to
* #CMID_HI, the error represented by the reason code is defined
* by OpenMVS. If the contents of the high order two bytes lie
* outside the range, the error represented by the reason code
* is not an OpenMVS reason code.
#CMID_LO            EQU 0000    Low range
#CMID_HI            EQU 8447    High range
* Define equates for console cmd
CC_MAX_MSG_LENGTH   EQU 17850  Max Wto string length for SUs @DIC
CC_MAX_MSG_LENGTH_NONSU EQU 17780 Max Wto string length for nonSU @DIC
CC_MODIFY_BUFFER_LENGTH EQU 128  Length of Modify Buffer @PAC
CONSOLE_MODIFY      EQU 1    Service interrupted by Modify @PAC
CONSOLE_STOP        EQU 2    Service interrupted by Stop @PAC
* Define equates for server_init syscall ManagerType parameter
SRV_WORKMGR         EQU 1    Work Manager services requested @DGA
SRV_QUEUEMGR        EQU 2    Queue Mgr services requested @DGA
SRV_SERVERMGR        EQU 4    Server Mgr services requested @DGA
SRV_SERVERMGRDYNAMIC EQU 8    Server Mgr With Dynamic mngt @DSA
* Define equates for server_pwu syscall FcnCode parameter
SRV_PUT_NEWWRK      EQU 1    Put new work function requested @DGA
SRV_PUT_SUBWRK      EQU 2    Put sub work function requested @DGA
SRV_TRANSFER_WRK    EQU 4    Transfer work function requested @DGA
SRV_GET_WRK         EQU 8    Get work function requested @DGA
SRV_REFRESH_WRK     EQU 16    Refresh work fcn requested @DGA
SRV_END_WRK         EQU 32    End work function requested @DGA
SRV_DEL_ENC         EQU 64    Delete Enclave Fcn requested @DGA
SRV_DISCONNECT      EQU 128    Disconnect from WLM @PBA
SRV_DISCONNECT_COND EQU 256    Disconnect conditional from WLM @PCA
*                               EQU 512    Reserved for internal use @PGA
*                               See BPXZCONS @PGA
* Define equates for BPX1SLK syscall LockFcnCode parameter
SLK_OBTAIN          EQU 1    Obtain function request @DZA
SLK_OBTAIN_COND     EQU 2    Obtain conditional function req @DZA
SLK_INIT            EQU 4    Initialization function request @DZA
SLK_DESTROY         EQU 8    Destroy function request @DZA
SLK_RELEASE         EQU 16    Release function request @DZA
* Define equates for BPX1SLK syscall LockReqType parameter

```



```

SLK_NORMAL          EQU 1 Normal request type @DZA
SLK_ERRORCHECK      EQU 2 Errorcheck request type @DZA
SLK_RECURSIVE       EQU 4 Recursive request type @DZA
* Define equates for BPX1SLK syscall LockType parameter
SLK_EXCLUSIVE        EQU 1 Exclusive lock type @DZA
SLK_SHARED          EQU 2 Shared lock type @DZA
*
* Constants for BPX1PCT pfscntl
*
* Constants for BPXVRCAC - LFS Cache @DHA
PC#ADDFILE          EQU X'80000007' Filecache cmd '80000007'x @PDC
PC#DELETEFILE       EQU X'80000008' Filecache cmd '80000008'x @PDC
PC#PURGECACHE       EQU X'8000000A' Filecache cmd '8000000A'x @PDC
PC#REFRESHCACHE     EQU X'80000009' Filecache cmd '80000009'x @PDC
PC#SHUTDOWNFILESYS EQU X'8000000B' Soft Shutdown '8000000B'x @DVA
PC#PfsRecycle       EQU X'8000000C' Shutdown PFS for recycle
PC#PfsRestart       EQU X'8000000D' Restart the PFS
*
PC#SETIBMASYIO      EQU X'C0000006' SetIbm AsyncIO 'C0000006'x@PDC
PC#SETIBMOPTCMD     EQU X'C0000005' SetIBMOpt TCP 'C0000005'x@PDC
PC#ERRORTEXT        EQU X'C000000B' Get error text 'C000000B'x@DNA
*                               -1073741813 @DNA
PC#SYSNAMES         EQU X'C000000E' Get sysnames 'C000000C'x@Px
PC#TDNAMES          EQU X'C000000F' GET CINET TDNAMES @03A
PC#HFSSTATS         EQU X'C0000010' GET HFS Stats @PKA
PC#BRLMSRVR        EQU X'C0000011' GET brlm server name @PKA
PC#SFSDIAG          EQU X'80000012' Shared-FS Diagnose @DTA
PC#USERSIGNAL       EQU X'C0000013' Set Lost Locks Signal(@05)@06A
PC#DIRGETHOST       EQU X'C0000014' Directed GetHost @06A
PC#SHUTTINGDOWNFS  EQU X'C0000015' File system shutdown @E7A
PC#RECYCLEDONE      EQU X'C0000016' recycle is complete @EDA
* PC#GetVnodeToken - Allocate Vde & insert an open
PC#GETVNODETOKEN    EQU X'80000017' @F2A
* inet6_is_srcaddr() function @F3A
PC#ISSRCADDR        EQU X'C0000018' inet6_is_srcaddr @F3A
PC#OWNEROK          EQU X'C000001C' owner query @POA
PC#MODIFYFYPFS      EQU X'C000001D' F OMVS,PFS= @F8A
PC#THREADUNDUB      EQU X'C000001E' Reserved for Thread undub @F9A
PC#EnablePgmAuth    EQU X'C000001F' Enable Program Authorization.
*                               Services will bypass user
*                               identity auth check if
*                               invoked by authorized pgm.
*                               Invoking program must be
*                               authorized to use. @09A
PC#DisablePgmAuth    EQU X'C0000020' Disable Program Authorization.
*                               Invoking program does not
*                               need to be authorized. @09A
PC#RENAMEFS          EQU X'C0000021' Rename filesystem @FAA
PC#EnableFileAuth    EQU X'C0000025' Enable File Authorization.
*                               Services will be treated as
*                               a superuser when performing
*                               auth check for files if
*                               invoked by authorized pgm.
*                               Invoking program must be
*                               authorized to enable. @0BA
PC#DisableFileAuth    EQU X'C0000026' Disable
*                               File Authorization.
*                               Invoking program does not
*                               need to be authorized. @0BA
PC#STOPPFS           EQU X'C0000031' notify the pfs a STOPPFS @FGA
*                               was issued
* PC#ErrorText Subfunctions - Request type:
PC#ETDESC            EQU X'0000' Get description text @DNA
PC#ETACTION          EQU X'0001' Get action text @DNA
PC#ETMODNAME         EQU X'0002' Get module name @DNA
* PC#ErrorText Error code type
PC#ETREASON          EQU X'0000' Reason code input @DNA
PC#ETERRNO           EQU X'0001' Errno code input @DNA
* PFS specific component ID for reason codes
PC#COMPIDUNI         EQU X'E4' unicode conversion @F7A
* PC#ShuttingDownFS args
PC#TYPEFILESYS       EQU X'00000001' shutdown=filesystem @E7A
PC#TYPEFILEOWNER     EQU X'00000002' shutdown=fileowner @E7A
PC#TYPEOMVS          EQU X'00000003' omvs shutdown @E7A
*
* Constants for BPX1LOD - HFS load - options @EFC
*
LOD_ERROR_ST_EXLINK EQU X'80000000' Error if sticky/ext lnk @DRA
LOD_IGNORE_STICKY   EQU X'40000000' Skip sticky check @DRA
*
*
* Constants for BPX1LDX - extended HFS load - options @EFC

```

```

*               and directed load returned parameters
*
* LOD_DIRECTED      EQU X'20000000'   Directed loadhfs           @ECA
*
* DIRECTEDLOADRETURNEDPARMS DSECT ,           @ECA
* DIRECTEDLOADMODULELENGTH64 DS FD Directed Load Mod Len AMODE64 @ECA
*                               ORG DIRECTEDLOADMODULELENGTH64     @ECA
*                               DS F                               @ECA
* DIRECTEDLOADMODULELENGTH DS F   Directed Load Mod Len AMODE31 @ECA
*
* DIRECTEDLOADMODULESTART64 DS AD Directed Load Mod Start AMODE64 @ECA
*                               ORG DIRECTEDLOADMODULESTART64     @ECA
*                               DS F                               @ECA
* DIRECTEDLOADMODULESTART DS A   Directed Load Mod Start AMODE31 @ECA
*
* DIRECTEDLOADMODULEENTRYPT64 DS AD Directed Load Mod Entry AMODE64 @ECA
*                               ORG DIRECTEDLOADMODULEENTRYPT64   @ECA
*                               DS F                               @ECA
* DIRECTEDLOADMODULEENTRYPT DS A   Directed Load Mod Entry AMODE31 @ECA
*
* * Constants for BPX1DSO                                     @DPA
*
* SW_SIGDLV_ENABLE# EQU 1                                     @DPA
* SW_SIGDLV_DISABLE# EQU 2                                    @DPA
*
* * Define equates for BPX10SE syscall function_code parameter
* OSENV_GET EQU 1 get function                               @DQA
* OSENV_SET EQU 2 set function                               @DQA
* OSENV_UNSET EQU 4 unset function                           @DQA
* OSENV_PERSIST EQU 8 persist function                       @DQA
* OSENV_UNPERSIST EQU 16 unpersist function                  @DQA
* * Define equates for BPX10SE syscall Request_Flags parameter
* OSENV_WLM EQU 1 WLM Enclave membership                    @DQA
* OSENV_SECURITY EQU 2 pthread security environment          @DQA
*
* * Define equates for BPX1PQG syscall RequestType parameter
* THDQ_FREEZE EQU 2 Freezes the threads identified in
*               the THDQ Data List array (BPXYTHDQ)
*               @DUA
* THDQ_UNFREEZE_ALL EQU 8 Unfreezes all threads that are frozen
*               in the caller process                        @DUA
* THDQ_GET_STATE EQU 1 Retrieves the state data for the
*               threads identified in the THDQ data
*               list array or for all threads.
*               This value can only be specified with
*               THDQ_FREEZE                                  @DUA
*
* * Items for prctl() syscall subfunction, BPX1PRC           60@FOA
* PR_SET_PDEATHSIG EQU 1 Constant for SET_PDEATHSIG
* PR_GET_PDEATHSIG EQU 2 Constant for GET_PDEATHSIG
* PR_GET_DUMPABLE EQU 3 Constant for GET_DUMPABLE
* PR_SET_DUMPABLE EQU 4 Constant for SET_DUMPABLE
* PR_GET_UNALIGN EQU 5 Constant for GET_UNALIGN
* PR_SET_UNALIGN EQU 6 Constant for SET_UNALIGN
* PR_GET_KEEPCAPS EQU 7 Constant for GET_KEEPCAPS
* PR_SET_KEEPCAPS EQU 8 Constant for SET_KEEPCAPS
* PR_GET_FPEMU EQU 9 Constant for GET_FPEMU
* PR_SET_FPEMU EQU 10 Constant for SET_FPEMU
* PR_GET_FPEXC EQU 11 Constant for GET_FPEXC
* PR_SET_FPEXC EQU 12 Constant for SET_FPEXC
* PR_GET_TIMING EQU 13 Constant for GET_TIMING
* PR_SET_TIMING EQU 14 Constant for SET_TIMING
* PR_SET_NAME EQU 15 Constant for SET_NAME
* PR_GET_NAME EQU 16 Constant for GET_NAME
* PR_GET_ENDIAN EQU 19 Constant for GET_ENDIAN
* PR_SET_ENDIAN EQU 20 Constant for SET_ENDIAN
* PR_GET_SECCOMP EQU 21 Constant for GET_SECCOMP
* PR_SET_SECCOMP EQU 22 Constant for SET_SECCOMP
* PR_CAPBSET_READ EQU 23 Constant for CAPBSET_READ
* PR_CAPBSET_DROP EQU 24 Constant for CAPBSET_DROP
* PR_GET_TSC EQU 25 Constant for GET_TSC
* PR_SET_TSC EQU 26 Constant for SET_TSC
* PR_GET_SECUREBITS EQU 27 Constant for GET_SECUREBITS
* PR_SET_SECUREBITS EQU 28 Constant for SET_SECUREBITS
* PR_SET_TIMERSLACK EQU 29 Constant for SET_TIMERSLACK
* PR_GET_TIMERSLACK EQU 30 Constant for GET_TIMERSLACK
* PR_TASK_PERF_EVENTS_DISABLE EQU 31 Constant for
*               TASK_PERF_EVENTS_DISABLE
* PR_TASK_PERF_EVENTS_ENABLE EQU 32 Constant for
*               TASK_PERF_EVENTS_ENABLE
*
* PR_MCE_KILL EQU 33 Constant for MCE_KILL
* PR_MCE_KILL_GET EQU 34 Constant for MCE_KILL_GET

```

```

PR_SET_MM EQU 35 Constant for SET_MM
PR_SET_CHILD_SUBREAPER EQU 36 Constant for SET_CHILD_SUBREAPER
PR_GET_CHILD_SUBREAPER EQU 37 Constant for GET_CHILD_SUBREAPER
PR_SET_NO_NEW_PRIVS EQU 38 Constant for SET_NO_NEW_PRIVS
PR_GET_NO_NEW_PRIVS EQU 39 Constant for GET_NO_NEW_PRIVS
PR_GET_TID_ADDRESS EQU 40 Constant for GET_TID_ADDRESS
PR_SET_THP_DISABLE EQU 41 Constant for SET_THP_DISABLE
PR_GET_THP_DISABLE EQU 42 Constant for GET_THP_DISABLE
PR_MPX_ENABLE_MANAGEMENT EQU 43 Constant for
* MPX_ENABLE_MANAGEMENT
PR_MPX_DISABLE_MANAGEMENT EQU 44 Constant for
* MPX_DISABLE_MANAGEMENT
PR_SET_FP_MODE EQU 45 Constant for SET_FP_MODE
PR_GET_FP_MODE EQU 46 Constant for GET_FP_MODE
PR_CAP_AMBIENT EQU 47 Constant for CAP_AMBIENT
PR_SVE_SET_VL EQU 50 Constant for SVE_SET_VL
PR_SVE_GET_VL EQU 51 Constant for SVE_GET_VL
PR_GET_SPECULATION_CTRL EQU 52 Constant for GET_SPECULATION_CTRL
PR_SET_SPECULATION_CTRL EQU 53 Constant for SET_SPECULATION_CTRL
PR_PAC_RESET_KEYS EQU 54 Constant for PAC_RESET_KEYS
PR_SET_TAGGED_ADDR_CTRL EQU 55 Constant for SET_TAGGED_ADDR_CTRL
PR_GET_TAGGED_ADDR_CTRL EQU 56 Constant for GET_TAGGED_ADDR_CTRL
PR_SET_IO_FLUSHER EQU 57 Constant for SET_IO_FLUSHER
PR_GET_IO_FLUSHER EQU 58 Constant for GET_IO_FLUSHER
PR_SET_SYSCALL_USER_DISPATCH EQU 59 Constant for
* SET_SYSCALL_USER_DISPATCH
PR_PAC_SET_ENABLED_KEYS EQU 60 Constant for PAC_SET_ENABLED_KEYS
PR_PAC_GET_ENABLED_KEYS EQU 61 Constant for PAC_GET_ENABLED_KEYS
PR_SCHED_CORE EQU 62 Constant for SCHED_CORE
PR_SET_VMA EQU 1398164801 Constant for SET_VMA
PR_SET_PTRACER EQU 1499557217 Constant for SET_PTRACER
* Constants for __authenticate syscall 6@FTA
* __authenticate Auth_cred_type parameter constants
AUTH_USER_ID# EQU 1 Authenticate with userid
AUTH_ID_TOKEN# EQU 2 Authenticate with IDT
* __authenticate Option_flags parameter constants
AUTH_BUILD_IDT# EQU 1 Request to build an IDT
AUTH_RETURN_USERNAME# EQU 2 Return userid associated
* with IDT
AUTH_MSGRTRN# EQU 4 Return RACF messages
AUTH_RETURNED_IDT# EQU 8 A new or changed IDT was
* returned
*
** BPXYCONS End

```

## BPXYCW — Serialization constants used by many services

BPXYCW is composed only of EQUates. DSECT= is allowed but ignored.

```

          BPXYCW
** BPXYCW:   Serialization Constants
CW_INTRPT EQU 1 Thread interrupted by a signal
* (x'0000 0001')
CW_CONDVAR EQU 32 Thread notified that some condition
* has been met (x'0000 0020')
CW_TIMEOUT EQU 64 Timeout occurred (x'0000 0040')
*
** BPXYCW End

```

## BPXYDCOR — dbx cordump cache information

BPXYDCOR contains the mapping of dump related information used by dbx when a dump is being formatted. AMODE 64 callers use “BPXYDCOR — dbx cordump cache information” on page 1190.

```

          BPXYDCOR
*
* *****
* *
* * Level information
* *
* *****
*
*

```

```

DCOR_LEVEL1 EQU 65536      65536='00010000'x.
DCOR_LEVEL2 EQU 131072    131072='00020000'x.
*
* *****
* *
* * Function codes for BPXGMCDE routine
* *
* *****
*
*
DCOR_OPEN# EQU 1
DCOR_CLOSE# EQU 2
DCOR_STATUS# EQU 3
*
* *****
* *
* * Open return codes
* *
* *****
*
*
DCOR_CDERC_OK EQU 0      The specified function completed successfully
DCOR_CDERC_PARMERR EQU 4 A parameter error was detected. See return X
                        value 1 for more detail
DCOR_CDERC_PROCERR EQU 8 A DCORE processing error occurred. See return X
                        value 1 for more detail
DCOR_CDERC_IKJTSOEVERR EQU 12 An error was encountered trying to X
                        establish a TSO environment with the IKJTSOEV X
                        service. See return values for more X
                        information
DCOR_CDERC_IKJEFTSRERR EQU 16 An error was encountered trying to run X
                        the REXX EXEC with the IKJEFTSR service. See X
                        return values for more information
DCOR_CDERC_ALLOCATEERR EQU 20 An error was encountered trying to X
                        allocate one of the user specified data sets.
DCOR_CDERC_IRXINITERR EQU 28 An error was encountered trying to X
                        establish a REXX environment
*
* *****
* *
* * Status return codes
* *
* *****
*
*
DCOR_CDERC_STATUS_OPENCOMPLETE EQU 0
DCOR_CDERC_STATUS_OPENCONTINUING EQU 1
DCOR_CDERC_STATUS_OPENTERMINATED EQU 2
DCOR_CDERC_STATUS_INVALIDTOKEN EQU 3
*
* *****
* *
* * Status Rc values when Status return code is
* * * Dcor_CDErc_Status_OpenContinuing
* *
* *****
*
*
DCOR_STATUS_CONT_STARTTSOENV EQU 0 Starting the TSO environment
DCOR_STATUS_CONT_EXECSTARTED EQU 1 BPXTIPCS started
DCOR_STATUS_CONT_EXECCLIST EQU 2 BPXTIPCS allocating CLIST data set
DCOR_STATUS_CONT_DUMPDDIR EQU 3 BPXTIPCS allocating/creating dump X
                        directory via BLSCDDIR
DCOR_STATUS_CONT_ALLOCDUMPDS EQU 4 BPXTIPCS allocating the dump data X
                        set
DCOR_STATUS_CONT_INVOKEIPCS EQU 5 BPXTIPCS invoking IPCS
DCOR_STATUS_CONT_INVOKEVERBX EQU 6 BPXTIPC2 invoking VERBX routine
DCOR_STATUS_CONT_ANALYSISSTART EQU 7 Dump analysis started
DCOR_STATUS_CONT_ANALYSISPROCASIDS EQU 8 Analysis processing Asids
DCOR_STATUS_CONT_EXECEXITING EQU 9 BPXTIPCS exiting
DCOR_STATUS_CONT_RECALL EQU 10 BPXTIPCS recalling data set
*
* *****
* *
* * R1 values when return code is Dcor_CDErc_ParmErr
* *
* *****
*
*
DCOR_R1_PARMERR_DUMPDSNREQ EQU 1 The name of a dump data set is X
                        required
DCOR_R1_PARMERR_HFSDSNREQ EQU 2 The name of a dump data set in the HFS X

```

```

could not be found

*
* *****
* *
* * R1 values when return code is Dcor_CDErc_ProcErr *
* *
* *****
*
*
DCOR_R1_PROCERR_SYSTEMERRATC EQU 1 An unexpected system error has      X
                                occurred while trying to establish the IPCS X
                                environment. The R2 value contains an ABEND X
                                reason code
*
* *****
* *
* * R1 values when return code is Dcor_CDErc_AllocateErr *
* *
* *****
*
*
DCOR_R1_ALLOCATEERR_LOGDSN EQU 1 Error allocating the log data set.      X
                                The R2 field is the return code from      X
                                allocation and the R3 field is the reason  X
                                code.
DCOR_R1_ALLOCATEERR_EXECDN EQU 2 Error allocating the EXEC data set.      X
                                The R2 field is the return code from      X
                                allocation and the R3 field is the reason  X
                                code.
*
* *****
* *
* * Function codes for BPXGMPTR Ptrace Dump Access Routine *
* *
* *****
*
*
DCOR_ASID_LIST# EQU 1
DCOR_SET_ASID# EQU 2
DCOR_PID_LIST# EQU 3
DCOR_SET_PID# EQU 4
DCOR_LDINFO# EQU 5
DCOR_THREAD_LIST# EQU 6
DCOR_THREAD_CURRENT# EQU 7
DCOR_SET_THREAD# EQU 8
DCOR_PSW# EQU 9
DCOR_GPR_LIST# EQU 10
DCOR_THREAD_STATUS# EQU 11
DCOR_READ_D# EQU 12
DCOR_ERROR_PSW# EQU 13
DCOR_CAPTURE# EQU 14
DCOR_ERROR_GPR_LIST# EQU 15
DCOR_FLT_LIST# EQU 16
DCOR_ERROR_FLT_LIST# EQU 17
DCOR_CONDINFO# EQU 18
DCOR_IPCSCMD# EQU 19
DCOR_PTRRC_OKVALUE EQU 0 The specified function completed successfully
DCOR_PTRRC_ASIDNOTFOUND EQU 1 The requested asid(s) not in dump
DCOR_PTRRC_ASIDNOTSET EQU 2 An ASID or PID has not been established      X
                                for this session
DCOR_PTRRC_REQTYPE_NOTDEFINED EQU 3 The function type provided on this    X
                                request is not supported by BPXGMPT2
DCOR_PTRRC_REQINVALIDTOKEN EQU 4 The open token provided on this          X
                                request is not valid
DCOR_PTRRC_REQDCORTERMINATED EQU 5 Dcor dump access services are not      X
                                available
DCOR_PTRRC_THREADNOTFOUND EQU 6 The request thread(s) were not in the      X
                                dump
DCOR_PTRRC_THREADNOTSET EQU 7 The current thread has not been              X
                                established
DCOR_PTRRC_PIDNOTSET EQU 9 The request PID(s) were not in the dump
DCOR_PTRRC_PIDNOTFOUND EQU 10 The current process has not been            X
                                established
DCOR_PTRRC_STORAGE_NOTINDUMP EQU 11 The requested storage was not          X
                                dumped
DCOR_PTRRC_NASTANDALONEDUMP EQU 12 Not supported in a standalone dump
DCOR_PTRRC_ABENDOCURRED EQU 13 Not supported in a standalone dump
DCOR_PTRRC_STORAGE_LENGTH_BAD EQU 14 The requested storage length was      X
                                zero
DCOR_PTRRC_SOME_STORAGE_INDUMP EQU 15 The number of bytes of storage        X
                                successfully retrieved is returned in the  X
                                reason code field

```

```

RSNOKVALUE EQU 0
RSNDCORERROR EQU 1      See Dcor return codes
RSNMVSEERROR EQU 2      Usually an out of storage condition or an    X
                           abend
RSNIPCSERROR EQU 3      When An IPCS error occurs use the DCOR log to X
                           view the messages generated by IPCS (normally X
                           suppress)

RSNCSVERROR EQU 4
RSNCSVMODI12ERR EQU 1
RSNCSVMODI3ERR EQU 2
RSNCSVT00MANYEXTENTS EQU 3
*
* *****
* * parameter definitions for BPXGMPTR Ptrace Dump Access Routine *
* * 1. Parm 1 function code *
* * 2. Parms 2 Token returned from DCOR_OPEN# *
* * 3. Parms 3-5 Function parameters *
* * 3. Parms 6-8 retvalue, retcode, rsncode *
* *****
*
*
PARMS DSECT
PARMS_FUNCYPEPTR DS 1AL4
PARMS_DCOMTOKENPTR DS 1AL4
PARMS_INTERFACE DS 0CL0012
        ORG PARMS_INTERFACE
PARMS_CAPTURE DS 0CL0012
PARMS_CAPTURE_PSTORADR DS 1AL4
PARMS_CAPTURE_PSTORLEN DS 1AL4
PARMS_CAPTURE_PDATABADR DS 1AL4 Address output buffer
        ORG PARMS_INTERFACE
PARMS_READD DS 0CL0012
PARMS_READD_PSTORADR DS 1AL4
PARMS_READD_PSTORLEN DS 1AL4
PARMS_READD_PDATABADR DS 1AL4 user provided buffer
        ORG PARMS_INTERFACE
PARMS_LDINFO DS 0CL0004
PARMS_LDINFO_OUTBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_THREADLIST DS 0CL0008
PARMS_THREADLIST_OUTBUFPTR DS 1AL4
PARMS_THREADLIST_OUTBUFCNT DS 1AL4
        ORG PARMS_INTERFACE
PARMS_PIDLIST DS 0CL0008
PARMS_PIDLIST_OUTBUFPTR DS 1AL4
PARMS_PIDLIST_OUTBUFCNT DS 1AL4
        ORG PARMS_INTERFACE
PARMS_ASIDLIST DS 0CL0008
PARMS_ASIDLIST_OUTBUFPTR DS 1AL4
PARMS_ASIDLIST_OUTBUFCNT DS 1AL4
        ORG PARMS_INTERFACE
PARMS_THREADCURRENT DS 0CL0004
PARMS_THREADCURRENT_OUTBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_SETASID DS 0CL0004
PARMS_SETASID_INBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_SETPID DS 0CL0004
PARMS_SETPID_INBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_SETTHREAD DS 0CL0004
PARMS_SETTHREAD_INBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_PSW DS 0CL0004
PARMS_PSW_OUTBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_ERROR_PSW DS 0CL0004
PARMS_ERROR_PSW_OUTBUFPTR DS 1AL4
        ORG PARMS_INTERFACE
PARMS_THREADSTATUS DS 0CL0008
PARMS_THREADSTATUS_OUTBUFPTR DS 1AL4
PARMS_THREADSTATUS_OUTBUFLLEN DS 1AL4
        ORG PARMS_INTERFACE
PARMS_GPRLIST DS 0CL0008
PARMS_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_GPRLIST_OUTBUFLLEN DS 1AL4
        ORG PARMS_INTERFACE
PARMS_ERROR_GPRLIST DS 0CL0008
PARMS_ERROR_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_GPRLIST_OUTBUFLLEN DS 1AL4
        ORG PARMS_INTERFACE
PARMS_FLTLIST DS 0CL0008

```

```

PARMS_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_FLTLIST_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_ERROR_FLTLIST DS 0CL0008
PARMS_ERROR_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_FLTLIST_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_CONDITIONINFO DS 0CL0008
PARMS_CONDITIONINFO_OUTBUFPTR DS 1AL4
PARMS_CONDITIONINFO_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_IPCSCMD DS 0CL0012
PARMS_IPCSCMDTEXT_INBUFPTR DS 1AL4
PARMS_IPCSCMDTEXT_INBUFLN DS 1AL4
PARMS_IPCSCMDPRNT_LRECL DS 1AL4
PARMS_XRVPTR DS 1AL4      Return Value
PARMS_XRCPTR DS 1AL4      Return Code
PARMS_XRSNPTR DS 1AL4     Reason Code
PARMS_LEN EQU *-PARMS
PARMSG DSECT
PARMS_FUNCYPEPTRG DS 1AD
PARMS_DCOMTOKENPTRG DS 1AD
PARMS_INTERFACEG DS 0CL0024
    ORG    PARMS_INTERFACEG
PARMS_CAPTUREG DS 0CL0024
PARMS_CAPTURE_PSTORADRG DS 1AD
PARMS_CAPTURE_PSTORLENG DS 1AD
PARMS_CAPTURE_PDATAADRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_READDG DS 0CL0024
PARMS_READD_PSTORADRG DS 1AD
PARMS_READD_PSTORLENG DS 1AD
PARMS_READD_PDATAADRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_LDINFOG DS 0CL0008
PARMS_LDINFO_OUTBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_THREADLISTG DS 0CL0016
PARMS_THREADLIST_OUTBUFPTRG DS 1AD
PARMS_THREADLIST_OUTBUFCNTG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_PIDLISTG DS 0CL0016
PARMS_PIDLIST_OUTBUFPTRG DS 1AD
PARMS_PIDLIST_OUTBUFCNTG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_ASIDLISTG DS 0CL0016
PARMS_ASIDLIST_OUTBUFPTRG DS 1AD
PARMS_ASIDLIST_OUTBUFCNTG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_THREADCURRENTG DS 0CL0008
PARMS_THREADCURRENT_OUTBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_SETASIDG DS 0CL0008
PARMS_SETASID_INBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_SETPIDG DS 0CL0008
PARMS_SETPID_INBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_SETTHREADG DS 0CL0008
PARMS_SETTHREAD_INBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_PSWG DS 0CL0008
PARMS_PSW_OUTBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_ERROR_PSWG DS 0CL0008
PARMS_ERROR_PSW_OUTBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_THREADSTATUSG DS 0CL0016
PARMS_THREADSTATUS_OUTBUFPTRG DS 1AD
PARMS_THREADSTATUS_OUTBUFLN DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_GPRLISTG DS 0CL0016
PARMS_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_GPRLIST_OUTBUFLN DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_ERROR_GPRLISTG DS 0CL0016
PARMS_ERROR_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_GPRLIST_OUTBUFLN DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_FLTLISTG DS 0CL0016
PARMS_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_FLTLIST_OUTBUFLN DS 1AD

```

```

        ORG    PARM_INTERFACEG
PARMS_ERROR_FLTLISTG DS 0CL0016
PARMS_ERROR_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_FLTLIST_OUTBUFLNG DS 1AD
        ORG    PARM_INTERFACEG
PARMS_CONDITIONINFOG DS 0CL0016
PARMS_CONDITIONINFO_OUTBUFPTRG DS 1AD
PARMS_CONDITIONINFO_OUTBUFLNG DS 1AD
        ORG    PARM_INTERFACEG
PARMS_IPCSCMDG DS 0CL0024
PARMS_IPCSCMDTEXT_INBUFPTRG DS 1AD
PARMS_IPCSCMDTEXT_INBUFLNG DS 1AD
PARMS_IPCSCMDPRNT_LRECLG DS 1AD
PARMS_XRVPTRG DS 1AD
PARMS_XRCPTRG DS 1AD
PARMS_XRSNPTRG DS 1AD
PARMSG_LEN EQU *-PARMSG
ASIDLIST_MAP DSECT
ASID_NEXTOFF DS 1FL4      Offset to the next ASID in DcomAsidList
ASID_NUM DS 1FL2
ASID_CPU DS 1FL1         CPUID
ASID_FLAGS DS 0BL1       Status flags
ASID_HOME EQU X'80'      Current HOMEAsid
ASID_PRIM EQU X'40'      Current PRIMARY ASID
ASID_SEC EQU X'20'       Current SECONDARY ASID
        ORG    ASID_FLAGS+X'00000001'
ASID_JOBNAME DS 1CL0009
        DS 1CL0003      Reserved
ASID_PIDCNT DS 1FL4       Number of Pids in this Asid
ASID_ASCB DS 1AL4         Pointer to ASCB
        DS 1CL0004      Reserved
ASID_PIDLISTPTR DS 1AL4   Pointer to the pidlist for This Asid
        DS 1CL0004      Reserved
ASID_MAPEND DS 0C         end of block
ASIDLIST_MAP_LEN EQU *-ASIDLIST_MAP
PIDLIST_MAP DSECT
PID_NEXTOFF DS 1FL4       Offset to the next Pid in DcomPidList
PID_ DS 1FL4              Process id
PID_ASID DS 1FL2          Asid of this Pid
PID_THIDCNT DS 1FL2       Count of thids in this pid
PID_FOCUSTHREAD DS 1CL0008 Ptrace focus thread
PID_ERRORTHREAD DS 1CL0008 Ptrace error thread
PID_LOGINNAME DS 1CL0009 Tso logon
        DS 1CL0003      Reserved
PID_THIDLISTPTR DS 1AL4   list info for each THID
        DS 1CL0004      Reserved
PID_PENDINGSIGMASK DS 1BL8 Signals pending at the process that could   X
                        not be delivered to any thread
PID_BLOCKEDSIGMASK DS 1BL8 Signals blocked on all thread
PID_MAPEND DS 0C         end of block
PIDLIST_MAP_LEN EQU *-PIDLIST_MAP
CONDINFO_MAP DSECT
COND_CURABENDINFO DS 0CL0016 If current task abended
COND_CURINTCODE DS 1FL2   Interrupt code
COND_CURSIGNUMBER DS 1FL2 Signal number raised
COND_CURABENDCODE DS 0BL4 Abend code
COND_CURABENDFLAGS DS 1BL1 System or user
COND_CURABENDCC DS 1BL3   Abend Number
COND_CURABENDREASON DS 1BL4 Abend Reason
COND_CURILC DS 1FL2       Instruction length
        DS 1CL0002      Reserved
CONDINFO_MAPEND DS 0C     end of block
CONDINFO_MAP_LEN EQU *-CONDINFO_MAP

```

## BPXYDIRE — Map directory entries for readdir

DSECT=NO is not allowed; the basing for the PFSOTHER data is not known, as it depends on the length of the name.

```

        BPXYDIRE
** BPXYDIRE: Mapping of directory entry
** Used By: RDD
* LA      RegOne,buffer          RegOne->BPX1RDD buffer and 1st DIRE
* USING   DIRE,RegOne           Addressability to DIRE
DIRE      DSECT ,
DIRENTINFO DS 0X               Fixed length information
DIRENTLEN DS H                 Entry length

```



```

DIRENTNAML      DS      H      Name length
DIRENTNAME      DS      0C     Name
* LR      RegTwo,RegOne      RegTwo->DIRE
* LA      RegTwo,4(RegTwo)    RegTwo->start of name
* SLR     RegThree,RegThree   Clear register
* ICM     RegThree,3,DIRENTNAML Load name length
* ALR     RegTwo,RegThree     RegTwo->end of name+1
* USING   DIRENTPFSDATA,RegTwo Addressability to DIRENTPFSDATA
DIRENTPFSDATA    DSECT ,      Physical file system-specific data
DIRENTPFSINO     DS      CL4   File Serial Number = st_ino
DIRENTPFSOTHER   DS      0C    Other PFS specific data
                ORG      DIRENTPFSDATA
DIRENTPLUSATTR   DS      0C    ReaddirPlus Attr
*
* ICM     RegThree,3,DIRENTLEN Load entry length
* ALR     RegOne,RegThree     RegOne->Next DIRE in buffer
* BCT     Return_Value,Back_to_process_next_DIRE
** BPXYDIRE End

```

## BPXYENFO – ENF signal constants

BPXYENFO is composed only of EQUates for listeners of kernel ENF signals. DSECT= is allowed but ignored.

```

                BPXYENFO      ,
** BPXYENFO: OMVS ENF constants
** Used By: OMVS ENF Listeners and OMVS ENF Signallers
* OMVS ENF QUALifier values
BPXYENFOACT EQU  X'80000000'    OMVS Active
** BPXYENFO End

```

## BPXYEFD— Map eventfd syscall parameters

```

EFD_SEMAPHORE EQU 8192
EFD_CLOEXEC EQU 4096
EFD_NONBLOCK EQU 4

```

## BPXYEPOL— Map epoll syscall parameters

```

                BPXYEPOL      ,
** BPXYEPOL: Epoll Events Control Block
** Used by: epoll I/O event notification facility
* *****
EPOLLEVENT      DSECT ,
EPOLL_ID        DS      CL8      Eye Catcher 'EPOLLEV '
EPOLLEVENTS     DS      CL4      I/O event identifiers
                ORG      EPOLLEVENTS
EPOLLEVENTS_INPUT DS      CL1
* Bit definitions:
EPOLL_ONESHOT   EQU  X'40'      Input FD disabled after Notify
EPOLL_EXCLUSIVE EQU  X'20'      Input -Notify actions
EPOLLEVENTS_B2 DS      CL1
                ORG      EPOLLEVENTS_B2
                DS      B      Unused
EPOLLEVENTS_B3 DS      CL1
                ORG      EPOLLEVENTS_B3
                DS      B      Unused
EPOLLEVENTS_INOUT DS      CL1      I/O Event Identifiers
* Bit definitions:
EPOLL_HUP      EQU  X'40'      Output -Hangup occurred
EPOLL_ERR      EQU  X'20'      Output -Error occurred
EPOLL_PRI      EQU  X'10'      In/Out Exceptional Condition
EPOLL_WRBAND   EQU  X'08'      In/Out Priority Data Write
EPOLL_OUT      EQU  X'04'      In/Out Write Operation Ready
EPOLL_IN       EQU  X'03'      In/Out Read Operation Ready
EPOLL_RDBAND   EQU  X'02'      In/Out non-Zero band Read
EPOLL_RDNORM   EQU  X'01'      In/Out Band 0 read
                ORG      EPOLLEVENTS+4
                DS      CL4      Available

```

```

EPOLL_DATA DS CL8
EPOLL_PTR DS A
EPOLL_FD DS F
EPOLL_U32 DS F
EPOLL_U64 DS FL8
*
* *****
* * Constants *
* *****
EPOLL_CLOEXEC EQU 1 epoll_create Option(1)
EPOLL_CTL_ADD EQU 0 epoll_ctl Option(1)
EPOLL_CTL_MOD EQU 1 epoll_ctl Option(2)
EPOLL_CTL_DEL EQU 2 epoll_ctl Option(3)
EPOLL#ID_0T03 EQU C'EPOL' This is the first 4-byte segment of an
8-byte constant. Eyecatcher
EPOLL#ID_4T07 EQU C'LEV ' This is the second 4-byte segment of an
8-byte constant. Eyecatcher
EPOLLEV#LEN EQU 24 Struct length
*
* Constant - EpollEvents area - User Mask Values for input only
*
EPOLLONESHOT EQU 1073741824
EPOLLEXCLUSIVE EQU 536870912
*
* Constant - EpollEvents area - User Mask Values for input/output
*
EPOLLHUP EQU 64
EPOLLERR EQU 32
EPOLLPRI EQU 16
EPOLLWRBAND EQU 8
EPOLLOUT EQU 4
EPOLLIN EQU 3
EPOLLRDBAND EQU 2
EPOLLRDNORM EQU 1
EPOLLEVENT_LEN EQU *-EPOLLEVENT

```

## BPXYERNO — Component return and reason codes

BPXYERNO is composed only of EQUates. DSECT= is allowed but ignored. Because the return codes and reason codes that are in this macro are in *z/OS UNIX System Services Messages and Codes*, the expansion of this macro is suppressed.

```

BPXYERNO LIST=NO
PUSH PRINT BPXYERNO: z/OS UNIX Component return/reason codes
PRINT OFF
POP PRINT

```

## BPXYFCTL — Command values and flags for fcntl

```

BPXYFCTL ,
** BPXYFCTL: File descriptor flags and command values
** Used By: FCT
FCTL DSECT ,
* External file descriptor flags
FCTLFDFL1 DS B
FCTLR01 EQU X'80' Reserved-DO NOT USE THIS BIT!
* FCTLFDFLAGS must never be < 0
FCTLFDFL2 DS B Reserved
FCTLFDFL3 DS B Reserved
FCTLFDFL4 DS B
FCTLCL0FORK EQU X'02' 1= close_on_fork
FCTLCL0EXEC EQU X'01' 1= close_on_exec
* Command value definitions
F_CVT DSECT , F_CONTROL_CVT section
FCVT_CMD DS F Sub-Command
SETCVTALL EQU 4 Unicode Services enabled
SETCVTAUTOALL EQU 5 Unicode Services enabled if AUTOCVT(ALL)
SETCVTOFF EQU 0 Set Off
SETCVTON EQU 1 Set On

```

```

SETAUTOCVTON      EQU    2      Set On if AUTOCVT(YES)
QUERYCVT          EQU    3      Query current mode
FCVT_PCCSID       DS      H      Program CCSID
FCVT_FCCSID       DS      H      File CCSID
*
F_DUPFD           EQU    0      Duplicate file descriptor
F_GETFD           EQU    1      Get file descriptor flags
F_SETFD           EQU    2      Set file descriptor flags
F_GETFL           EQU    3      Get file status flags
F_SETFL           EQU    4      Set file status flags
F_GETLK           EQU    5      Get record locking information
F_SETLK           EQU    6      Set record locking information
F_SETLKW          EQU    7      Set record locking information -
*                               wait if blocked
F_DUPFD2          EQU    8      Duplicate file descriptor, option 2
F_CLOSEFD         EQU    9      Close file descriptors
F_GETOWN          EQU    10     Get process id or process group
F_SETOWN          EQU    11     Set process id or process group
F_SETTAG          EQU    12     Set File Tag
F_CONTROL_CVT     EQU    13     Control conversion
F_DUPFD3          EQU    17     Duplicate FD: DUP2 + Flags
*                               DUP3 value definition
F_DUP3            DSECT,       F_DUP3 section
Fdup3_FD          DS      F      dup3() new file descriptor
Fdup3_Flags       DS      F      dup3() flags
Fdup3CloExec      EQU    X'00001000' 4096 = close_on_exec

*
FCTL#LENGTH      EQU    *-FCTL Length of this structure
AT_FDCWD          EQU    -100
FCTL#LENGTH      EQU    *-FCTL Length of this structure
** BPXYFCTL End

```

## BPXYFDUM – Logical file system dump parameter list

DSECT=YES is required.

```

                BPXYFDUM    DSECT=YES
** BPXYFDUM: FDUM - LFS dump list passed to PFS initialization
FDUM            DSECT ,
FDUMBEGIN       DS      0D
*
FDUMPHDRINFO    DS      0F
FDUMPENTS       DS      F      NUMBER OF ENTRIES
FDUMPID         DC      C'FDUM'  EYE CATCHER
FDUMPHRES1      DS      CL8     SPACE RESERVED FOR EXPANSION
*
FDUM#LENH       EQU    *-FDUMBEGIN
*
FDUMPDATA       DSECT ,
                DS      0F      ONE SET FOR EACH AREA TO DUMP
FDUMPSTOKEN     DS      CL8     STOKEN FOR DUMP
FDUMPRES1       DS      CL8     RESERVED
FDUMPSTART      DS      F      FIRST BYTE TO DUMP
FDUMPEND        DS      F      LAST BYTE TO DUMP
*
FDUM#LENENT     EQU    *-FDUMPDATA
*
* To access the FDUM header (dumpptr must be a copy of pfsi_dumpptr):
* L      RegOne,dumpptr          RegOne->pfsi_dumpents from BPXYPFSI
* USING  FDUM,RegOne            Addressability to FDUM
*
* To access the first FDUMPDATA:
* LR     RegTwo,RegOne           RegTwo->FDUM
* LA     RegTwo,FDUM#LENH(RegTwo) RegTwo->FDUMPDATA
* USING  FDUMPDATA,RegTwo        Addressability to FDUMPDATA fields
*
* To access the next FDUMPDATA:
* LA     RegTwo,FDUM#LENENT(RegTwo) RegTwo-> next FDUMPDATA
*
** BPXYFDUM End

```

## BPXYFLK— Applying or removing advisory locks

```

*/*****
*DCB   *LOCK_SH Fixed(31) Constant(1), /* Shared: 0001'b          */
*LOCK_EX Fixed(31) Constant(2), /* Exclusive: 0010'b          */
*LOCK_UN Fixed(31) Constant(8), /* Unlock: 1000'b          */
*LOCK_NB Fixed(31) Constant(4); /* Non-Block: 0100'b          */
*/*****

```

## BPXYFTYP — File type definitions

BPXYFTYP is composed only of EQUates. DSECT= is allowed but ignored.

```

                BPXYFTYP
** BPXYFTYP: File type definitions
** Used By: FST MKD MKN OPN
FT_DIR          EQU    1      Directory File
FT_CHARSPEC     EQU    2      Character Special File
FT_REGFILE      EQU    3      Regular File
FT_FIFO         EQU    4      Named Pipe (FIFO) File
FT_SYMLINK      EQU    5      Symbolic link
*
FT_SOCKET       EQU    7      Socket File
*
** File format definitions (for chattr)
FTFFNA          EQU    0      Not specified
FTFFBINARY      EQU    1      Binary data
*
FTFFNL          EQU    2      Text data delimiters:
FTFFCR          EQU    3      New Line
FTFFLF          EQU    4      Carriage Return
FTFFCRLF        EQU    5      Line Feed
FTFFLFCR        EQU    6      CR & LF
FTFFCRNL        EQU    7      LF & CR
*
FTFFRECORD      EQU    8      CR & NL
*
** BPXYFTYP End

```

## BPXYFUIO — Map file system user I/O block

BPXYFUIO is used to map the user and file system I/O block.

```

                BPXYFUIO
** BPXYFUIO: User I/O block
** Used By: VRW VRD VRA
FUIO            DSECT
FUIOBEGIN       DS      0D
FUIOHDR         DS      0D
*
FUIOID          DC      C'FUIO'
FUIOLEN         DC      EBCDIC ID - FUIO
FUIOINFO        DS      0D      Length of this FUIO
*
FUIOBUFFERADDR64 DS      0CL8   Note: The following fields must
FUIOBUFFERADDR64P DS      0D     map to BPXZDDPL
FUIOBUFFERADDR   DS      F      64 Bit Real Buffer address
*
FUIOBUFFALET     DS      F      64 Bit Real Buffer address
FUIOCURSOR       DS      0F     Buffer address for READ or
FUIOCUR1         DS      F      WRITE, etc. Address of iov
FUIOCUR2         DS      F      for READV and WRITEV
FUIOIBYTESRW     DS      F      Alet associated with Buffer
FUIOASID         DS      H      Current position in the file
*
FUIOFLAGS        DS      XL1    Word 1 of cursor
*
FUIOIBYTESRW     DS      F      Word 2 of cursor
FUIOIBYTESRW     DS      F      Num of bytes to read or write
FUIOIBYTESRW     DS      F      (or iovcnt for READV and WRITEV)
FUIOASID         DS      H      Address Space ID
*
FUIOFLAGS        DS      XL1    Flags

```

```

*
FUIORWIND      EQU    FUIOFLAGS                      X
                                Indicates if READ or WRITE      X
                                0 - Read, 1 - Write
FUIO#RD        EQU    X'7F'    Read: AND with FUIORWIND
FUIO#WRT       EQU    X'80'    Write: OR with FUIORWIND
*
FUIOPSWKEY     EQU    FUIOFLAGS                      X
                                Describes bits 1 through 4 of
                                byte FUIOFLAGS                X
FUIOPSWKEYMASK EQU    X'78'    AND with FUIOPSWKEY to clear
                                non-PSWKEY bits in FUIOFLAGS    X
*
FUIOSYNC       EQU    X'04'    Sync on write requested
FUIOSYNCDONE   EQU    X'02'    Sync on write was done
FUIOCHKACC     EQU    X'01'    Perform access checking
FUIOFLAG2      DS      XL1     More flags
FUIOREALPAGE   EQU    X'80'    Real page address provided
FUIOLIMITEX    EQU    X'40'    File size limit exceeded
FUIOIOVINUIO   EQU    X'20'    uio contains an iov struc
FUIOSHUTD     EQU    X'10'    Do shutdown after send
FUIOADDR64     EQU    X'08'    64 bit addressing
FUIOVSPECIFIC  DS      CL8     Vnop Specific Fields
FUIOFSSIZELIMIT DS      0CL8    Rlimit support
FUIOFSSIZELIMITHW DS      F      hiword - filesize limit
FUIONONNEWFILES EQU    X'80'    can't create new files
FUIOFSSIZELIMITLW DS      F      loword - filesize limit
FUIOREL2SIZE   DS      0F      Fuio before Rel 3 expansion
*
FUIOINTERNAL   DS      0CL16
FUIOCURRBUFFPTR DS      F      Buffer currently being processed
FUIOCURRBUFFLEN DS      F      Length of current buffer
FUIOCURRBUFFOFFSET DS      F      Offset into current buffer
FUIOCURRIOENTRY DS      F      Iov entry being processed
*
FUIOIOVRESIDUALCNT DS      F      Num bytes remaining in iov str
FUIOTOTALBYTESRW DS      F      Total number of bytes to be moved
                                If FuioIovinUio=on,
                                this is the sum of all bytes
                                in the iov. Otherwise, this is
                                the same as FuioIBytesRW
                                X
                                X
                                X
                                X
FUIOBUFF64VADDR DS      D      64 Bit Virtual Buffer address
FUIOEND        DS      0F      End of FUIO
*-----
*   ReadDir Specific Information
*-----
FUIOREADDIR    ORG      FUIOVSPECIFIC
FUIORDINDEX    DS      F      Readdir Index
FUIORDDFLAGS   DS      0XL4    Readdir flags
                                DS      XL3
FUIORDDFLAGS4  DS      XL1     Readdir flags:
FUIOCVERRET    EQU    X'02'    Cookie Verifier Returned
FUIORDDPLUS    EQU    X'01'    ReaddirPlus requested
*-----
*   VN_ReadWriteV and VN_SRMsg Specific Information
*-----
FUIOSOCKETALETS ORG      FUIOVSPECIFIC
FUIOIOVALET    DS      F      SRMsg IOV Alet
FUIOIOVBUFFALET DS      F      All IOV's Buff's Alet
*
*   Readdir and ReaddirPlus Output Cookie Verifier
FUIOCVER       ORG      FUIOINTERNAL
                                DS      CL8    Cookie Verifier
                                ORG
*
*   Constants
*
FUIO#LEN       EQU    FUIOEND-FUIOBEGIN                X
                                Length of FUIO
FUIO#LENGTH    EQU    FUIO#LEN Length of FUIO
FUIO#REL2LEN   EQU    FUIOREL2SIZE-FUIOBEGIN            X
                                Length of Release 2 FUIO
FUIO#SP        EQU    3      Subpool for the FUIO
** BPXYFUIO End

```

## BPXYGIDN — Map data returned for getpwnam and getpwuid

DSECT=NO is not allowed. The storage belongs to the service and a pointer is returned to the invoker.

```

        BPXYGIDN
** BPXYGIDN: getpwnam, getpwuid amd getpwent return structure
** Used By: GPN GPU GPE
GIDN      DSECT ,      USING on GPN, GPU, GPE Return_value
GIDN_U_LEN DS    F      Length of GIDN_U_NAME      1-8
GIDN_U_NAME DS    0C     User name (trailing blanks)
* Add GIDN_U_LEN to Index or base to access next field
DS    F      Length of user ID      4
GIDN_USERID DS    F      User ID
DS    F      Length of group ID      4
GIDN_GROUPID DS    F      Group ID
GIDN_D_LEN DS    F      Length of GIDN_D_NAME      0-1023
GIDN_D_NAME DS    0C     Initial working directory name
* Add GIDN_D_LEN to Index or base to access next field
GIDN_P_LEN DS    F      Length of GIDN_P_NAME      0-1023
GIDN_P_NAME DS    0C     Initial user program name
GIDN#LENGTH EQU    *-GIDN Length less U_LEN, D_LEN and P_LEN
** BPXYGIDN End

```

## BPXYGIDS — Map data returned for getgrnam and getgrpid

DSECT=NO is not allowed. The storage belongs to the service and a pointer is returned to the invoker.

```

        BPXYGIDS
** BPXYGIDS: getgrnam, getgrgid and getgrent return structure
** Used By: GGI GGN GGE
GIDS      DSECT ,
GIDS_G_LEN DS    F      Length of GIDS_G_NAME      1-8
GIDS_G_NAME DS    0C     Group name (trailing blanks)
* Add GIDS_G_LEN to index or base to access following fields
DS    F      Length of group ID, always 4
GIDS_GROUPID DS    F      Group ID
GIDS_COUNT DS    F      Count of array elements
* Make a local copy of GIDS_COUNT
* Test: if local copy of GIDS_COUNT zero, quit
GIDS_M_LEN DS    F      Length of GIDS_M_NAME      1-8
GIDS_M_NAME DS    0C     Member name (trailing blanks)
* Add GIDS_M_LEN+4 to index or base
* Decrement local copy of GIDS_COUNT, goto test.
GIDS#LENGTH EQU    *-GIDS Length less all variable fields
** BPXYGIDS End

```

## BPXYGRD — Map the response structure for getrandom

```

*   Constant - User Mask Values
*
*
GRND_NONBLOCK EQU 1
GRND_RANDOM EQU 2

```

## BPXYINFY— Map inotify\_init syscall parameters

```

*
* %DCL BPXYINFY_INCLUDED CHAR EXT;
* %DEACTIVATE BPXYINFY_INCLUDED;
* %IF BPXYINFY_INCLUDED = '' %THEN
* %DO;
**/* Macro made bi-lingual on 21055. CBGEN compile date 20027 */
**/* /*
        MACRO
        BPXYINFY &DSECT=YES,&LIST=YES,&TITLE=YES,&INOTIFY_EVENT=YES
        GBLC &BPXYINFY_INCLUDED
        GBLC &ZCBPRINT
&BPXYINFY_INCLUDED SETC 'YES'
        AIF ('&LIST' EQ 'NONE').P0
        AIF ('&TITLE' EQ 'NO').P5
        TITLE 'BPXYINFY - INOTIFY event structure for syscall'

```

```

.P5      ANOP
**/ BPXYINFY 1;;
*/****START OF SPECIFICATIONS*****
*   $MAC (BPXYINFY) COMP(SCPX1) PROD(BPX):
*01* MACRO NAME: BPXYINFY
*01* DSECT NAME: N/A
*01* DESCRIPTIVE NAME: INOTIFY event structure for syscall
*02* ACRONYM: INFY
**/
*/*01* PROPRIETARY STATEMENT= */
*/****PROPRIETARY_STATEMENT***** */
*/* */
*/* LICENSED MATERIALS - PROPERTY OF IBM */
*/* 5655-ZOS COPYRIGHT IBM CORP. 2023 */
*/* */
*/* STATUS= HBB77D0 */
*/* */
*/****END_OF_PROPRIETARY_STATEMENT***** */
*/*
*
*01* EXTERNAL CLASSIFICATION: PI
*01* END OF EXTERNAL CLASSIFICATION:
*
*01* FUNCTION: Define inotify structures
*
*
*
*01* METHOD OF ACCESS:
*02* PL/X:
*
*   %INCLUDE SYSLIB(BPXYINFY)
*   By default, the INOTIFY is based on INFYPTR.
*   If other basing is desired, use %INOTIFYBASE='BASED(XXXXXX)'.
*   If %INOTIFYBASE is defaulted, a declare for INFYPTR is
*   generated.
*
*   If VIA basing is required, specify %INOTIFYVIA='VIA(YYYYYY)'.
*   If %INOTIFYVIA='VIA(INOTIFYALET)' is specified, a declare for
*   INOTIFYALET is generated.
*02* ASM:
*   With DSECT=NO, storage is allocated in line
*   and addressability is provided thru that DSECT or CSECT.
*
*   With DSECT=YES, a DSECT is produced and "USING INFYFD,reg"
*   is required for addressability. Here "reg" contains the
*   address of inotify()EV#LENGTH bytes of storage.
*   The default is DSECT=YES.
*
*01* SIZE: INOTIFYEV#LENGTH
*   INOTIFY_EVENT -- X'0010' bytes
*
*01* POINTED TO BY: N/A
*
*01* CREATED BY: invoker of inotify()
*
*01* DELETED BY: invoker of inotify()
*
*01* STORAGE ATTRIBUTES:
*02* SUBPOOL: N/A
*02* KEY: N/A
*02* RESIDENCY: Primary address space
*
*01* FREQUENCY:
*01*
*
*01* SERIALIZATION: None
*
*01* DEPENDENCIES: None
*
*
*01* COMPONENT: z/OS UNIX (SCPX1)
*
*01* DISTRIBUTION LIBRARY: AMACLIB
*
*01* EYE-CATCHER: None
*02* OFFSET:
*02* LENGTH:
*
*01* CHANGE-ACTIVITY:
*
* $D0=WI402735, HBB77D0, 210224, PETN: inotify() Initial version
*

```

```

****END OF SPECIFICATIONS*****/
*% /*
.P0      ANOP
        AIF  ('&ZCBPRINT' EQ 'NO').P1
        AIF  ('&LIST' EQ 'YES').P2
.P1      ANOP
        PUSH PRINT
        PRINT OFF
.P2      ANOP
        AIF  ('&INOTIFY_EVENT' EQ 'NO').L0002
        AIF  ('&DSECT' EQ 'YES').L0003
        AIF  ('&INOTIFY_EVENT' NE 'YES').L0002
.L0003   ANOP
*
* *****
* * Start of inotify() Events control block *
* *****
*
*
        AIF  ('&DSECT' EQ 'YES').L0004
INOTIFY_EVENT DS 0F
        AGO  .L0005
.L0004   ANOP
INOTIFY_EVENT DSECT
.L0005   ANOP
INOTIFY_FD DS F
INOTIFY_EVENTS DS BL4      Event mask
INOTIFY_COOKIE DS F
INOTIFY_LENGTH DS F
INOTIFY_NAME DS 0C
*
* *****
* * Constants and error codes *
* *****
*
INOTIFYID_0T03 EQU C'INOT' This is the first 4-byte segment of an 8-byte constant. Eyecatcher *
INOTIFYID_4T07 EQU C'IFY ' This is the second 4-byte segment of an 8-byte constant. Eyecatcher *
INOTIFYEV#LEN EQU 16      Struct
IN_NONBLOCK EQU 4         NonBlock
IN_CLOEXEC EQU 2          Close Exec
*
* The following are events that user-space can watch for using
* the inotify_add_watch system call.
*
*
IN_ACCESS EQU X'00000001' File was accessed
IN_MODIFY EQU X'00000002' File was modified
IN_ATTRIB EQU X'00000004' Metadata changed
IN_CLOSE_WRITE EQU X'00000008' Metadata changed
IN_CLOSE_NOWRITE EQU X'00000010' Unwritable close
IN_OPEN EQU X'00000020' File was opened
IN_MOVED_FROM EQU X'00000040' File moved from X
IN_MOVED_TO EQU X'00000080' File moved to Y
IN_CREATE EQU X'00000100' Subfile created
IN_DELETE EQU X'00000200' Subfile deleted
IN_DELETE_SELF EQU X'00000400' Self was deleted
IN_MOVE_SELF EQU X'00000800' Self was moved
*
* The following are events that are sent as needed to any watch.
*
*
IN_UNMOUNT EQU X'00002000' Fs unmounted
IN_Q_OVERFLOW EQU X'00004000' Event queued ovrfl
IN_IGNORED EQU X'00008000' File was ignored
*
* Helper events
*
*
IN_CLOSE EQU X'00000018' In close events
IN_MOVE EQU X'000000C0' Move from or to
*
* Special flags
*
* Only watch the path if it is a directory
*
*
IN_ONLYDIR EQU X'01000000' Watch dir path
IN_DONT_FOLLOW EQU X'02000000' Dont follow sym link
*
* Exclude events on unlinked objects

```



```

*
*
IN_EXCL_UNLINK EQU X'04000000' Exclude events
IN_MASK_CREATE EQU X'10000000' Only create watches
*
*           Add to the mask of an already existing watch
*
*
IN_MASK_ADD EQU X'20000000' Add to mask
IN_ISDIR EQU   X'40000000' Event against dir
IN_ONESHOT EQU X'80000000' Only sent once
IN_ALL_EVENTS EQU X'00000FFF' All events
INOTIFY_EVENT_LEN EQU *-INOTIFY_EVENT
.L0002      ANOP
            AIF  ('&ZCBPRINT' EQ 'NO').P3
            AIF  ('&LIST' EQ 'YES').P4
.P3         ANOP
            POP  PRINT
.P4         ANOP
.P_EXIT     ANOP
            MEND
**/ BPXYINFY_2;;
* %BPXYINFY_INCLUDED = 'YES';
* %DCL ZCBPRINT CHAR EXT;
* %DEACTIVATE ZCBPRINT;
* %DCL BPXYINFY_LIST CHAR EXT;
* %DEACTIVATE BPXYINFY_LIST;
* %IF BPXYINFY_LIST = 'NO' |
*   ZCBPRINT = 'NO' %THEN
* %DO;
*   @LIST PUSH NOECHO;
*   @LIST NOASSEMBLE NOECHO;
*   @LIST OFF C NOECHO;
* %END;
/* Start of PL/X Source */
* %DCL ASABEGIN_AMODE CHAR EXTERNAL;
* %DCL PTR_64 CHAR EXTERNAL;
* %IF INOTIFYBASE= '' %THEN
* %DO;
*   %INOTIFYBASE = 'BASED(INFYPTR)'; ! Default basing to INFYPTR
* %END;
* %IF TRANSLATE(INOTIFYBASE)='BASED(INFYPTR)' %THEN
* %DO;
*   %IF ASABEGIN_AMODE='64' | PTR_64='YES'
*   %THEN
*   %DO;
*     DCL INFYPTR Ptr(64);
*   %END;
*   %ELSE
*   %DO;
*     DCL INFYPTR Ptr(31);
*   %END;
* %END;
* %IF TRANSLATE(INOTIFYVIA)='VIA(INOTIFYALET)' %THEN
* %DO;
*   DCL INOTIFYAlet FIXED(32);
* %END;
*/*****
/* Start of inotify() Events control block */
/******
*Declare
* 1 INOTIFY_Event INOTIFYBASE INOTIFYVIA,
* 3 INOTIFY_fd Fixed(31),
* 3 INOTIFY_Events Bit(32), ! Event mask
* 3 INOTIFY_Cookie Fixed(32),
* 3 INOTIFY_Length Fixed(32),
* 3 INOTIFY_Name CHAR(0);
*
*/*****
/* Constants and error codes */
/******
*DECLARE
* INOTIFYID Char(8) Constant('INOTIFY '), ! Eyecatcher
* INOTIFYEV#LEN Fixed(32) Constant(Length(INOTIFY_Event)),! Struct
*
* IN_NONBLOCK Fixed(32) Constant('00000004'X), !NonBlock
* IN_CLOEXEC Fixed(32) Constant('00000002'X), !Close Exec
*
/* The following are events that user-space can watch for using */
/* the inotify_add_watch system call. */
* IN_ACCESS BIT(32) Constant('00000001'X), !File was accessed
* IN_MODIFY BIT(32) Constant('00000002'X), !File was modified

```

```

* IN_ATTRIB      BIT(32) Constant('00000004'X), !Metadata changed
* IN_CLOSE_WRITE BIT(32) Constant('00000008'X), !Metadata changed
* IN_CLOSE_NOWRITE BIT(32) Constant('00000010'X), !Unwritable close
* IN_OPEN        BIT(32) Constant('00000020'X), !File was opened
* IN_MOVED_FROM  BIT(32) Constant('00000040'X), !File moved from X
* IN_MOVED_TO    BIT(32) Constant('00000080'X), !File moved to Y
* IN_CREATE      BIT(32) Constant('00000100'X), !Subfile created
* IN_DELETE      BIT(32) Constant('00000200'X), !Subfile deleted
* IN_DELETE_SELF BIT(32) Constant('00000400'X), !Self was deleted
* IN_MOVE_SELF   BIT(32) Constant('00000800'X), !Self was moved
*
**/ The following are events that are sent as needed to any watch. */
* IN_UNMOUNT     BIT(32) Constant('00002000'X), !Fs unmounted
* IN_Q_OVERFLOW  BIT(32) Constant('00004000'X), !Event queued ovrl
* IN_IGNORED     BIT(32) Constant('00008000'X), !File was ignored
*
**/ Helper events */
* IN_CLOSE       BIT(32) Constant('00000018'X), !In close events
* IN_MOVE        BIT(32) Constant('000000C0'X), !Move from or to
*
**/ Special flags */
**/ Only watch the path if it is a directory */
* IN_ONLYDIR     BIT(32) Constant('01000000'X), !Watch dir path
* IN_DONT_FOLLOW BIT(32) Constant('02000000'X), !Dont follow sym link
**/ Exclude events on unlinked objects */
* IN_EXCL_UNLINK BIT(32) Constant('04000000'X), !Exclude events
* IN_MASK_CREATE BIT(32) Constant('10000000'X), !Only create watches
**/ Add to the mask of an already existing watch */
* IN_MASK_ADD    BIT(32) Constant('20000000'X), !Add to mask
* IN_ISDIR       BIT(32) Constant('40000000'X), !Event against dir
* IN_ONESHOT     BIT(32) Constant('80000000'X), !Only sent once
* IN_ALL_EVENTS  BIT(32) Constant('00000FFF'X); !All events
**/ End of PL/X Source */
* %IF BPXYINFY_LIST = 'NO' |
*   ZCBPRINT = 'NO' %THEN
* %DO;
*   @LIST POP NOECHO;
* %END;
* %END;

```

## BPXYINHE – Spawn Inheritance Structure

AMODE 64 callers use “BPXYINHE – Spawn inheritance structure” on page 1195.

```

                BPXYINHE
** BPXYINHE: Inheritance Area
** Used By: spawn() callable service
INHE            DSECT
INHEBEGIN       DS      0D
*
INHEEYE         DC      C'INHE' Eye catcher
INHELENGTH      DC      AL2(INHE#LENGTH) X
                Length of this structure
INHEVERSION     DC      AL2(INHE#VER)
INHE#VER        EQU     3      Version of this structure
INHEFLAGS       DS      0BL4   Flags indicating contents of structure
INHEFLAGS0      DS      XL1    1st byte
INHESETPGROUP   EQU     X'80'  Set Process Group using INHEPGROUP
INHESETSIGMASK  EQU     X'40'  Set Signal Mask using INHESIGMASK
INHESETSIGDEF   EQU     X'20'  Set Signal Defaults using INHESIGDEF
INHESETTCPRGP  EQU     X'10'  Set TTY Pgrp using INHECTLTTFD
INHESETCWD      EQU     X'08'  Set CWD using INHECWDPTR
INHESETUMASK    EQU     X'04'  Set UMASK using INHEUMASK
INHESETUSERID   EQU     X'02'  Set Userid using INHEUSERID
INHESETREGIONSZ EQU     X'01'  Set Region using INHEREGIONSZ
INHEFLAGS1      DS      XL1    2nd byte
INHESETTIMELIMIT EQU     X'80'  Set Timelimit with INHETIMELIMIT
INHESETACCTDATA EQU     X'40'  SET ACCTDATA using INHEACCTDATA
INHESETJOBNAME  EQU     X'20'  SET JOBNAME using INHEJOBNAME
INHEMUSTBELOCAL EQU     X'10'  Spawn locally or else fail
INHESETDEBUGENV EQU     X'08'  Setup Debug Environment
INHESETMEMLIMIT EQU     X'04'  Set MemLimit with INHEMEMLIMIT
INHEFLAGS2      DS      XL1    3rd byte
INHEFLAGS3      DS      XL1    4th byte
INHEPGROUP      DS      F      Process Group for child
INHE#NEWPGROUP  EQU     0      Put child in a new proc grp of its own
INHESIGMASK     DS      BL8    Signal Mask for child
INHESIGDEF      DS      BL8    Set of default signals for child

```

```

INHECTLTTFD      DS    F    Cntl TTY FD for tcsetgrp() in child
INHECWDPTR       DS    0F    31-Bit Addressing Version
INHECWDLEN       DS    F    Pointer to the users CWD
INHEACCTDALEN    DS    H    Length of the users CWD
INHEACCTDATALEN  DS    H    LENGTH OF THE USERS ACCTDATA
INHEACCTDATAPTR  DS    F    POINTER TO THE USERS ACCTDATA
INHEUMASK        DS    XL4   Users Umask
INHEUSERID       DS    CL8   New A.S. user identity
INHEJOBNAME      DS    CL8   New A.S. jobname
INHEREGIONSZ     DS    F    New A.S. region size
INHETIMELIMIT    DS    F    New A.S. Time limit
                 DS    CL20  reserved
INHEMMLIMIT      DS    D    New A.S. Memlimit #bytes
INHE#LENGTH      EQU    *-INHE
** BPXYINHE End

```

## BPXYIOCC – Ioctl command definitions

BPXYIOCC is composed only of EQUates. DSECT= is allowed but ignored.

```

BPXYIOCC
** BPXYIOCC: Ioctl Command Constant Definitions
** Used By: Ioctl syscalls
* Ioctl command constants - Range 1-255 reserved for OpenMVS
* Authorized/Tcpip CMD values
IOCC#TCI          EQU    5000 Cmd for Tcpip Initialization
IOCC#TCC          EQU    5001 Cmd for Complete Tcpip Initialization
IOCC#TCS          EQU    5002 Cmd for Tcpip Path Sever
IOCC#TCR          EQU    5003 Cmd for Tcpip Reply/Post call
IOCC#TCG          EQU    5004 Cmd for Tcpip Signal call @p3a
IOCC#TCCE         EQU    5006 Cmd for Tcpip End Registration @D5A
SIOCMSDELRT       EQU    5007 Cmd for Delete Route
*                (Pre-Router wrap) @D5A
SIOCMSADDRT       EQU    5008 Cmd for Add Route
*                (Pre-Router wrap) @D5A
SIOCMSIFADDR      EQU    5009 Cmd for Set Interface address
*                (Pre-Router wrap) @D5A
SIOCMSIFFLAGS     EQU    5010 Cmd for Set Interface Flags
*                (Pre-Router wrap) @D5A
SIOCMSIFDSTADDR   EQU    5011 Cmd for Set point-to-point interface
*                address (Pre-Router wrap) @D5A
SIOCMSIFBRDADDR   EQU    5012 Cmd for Set Broadcast address
*                (Pre-Router wrap) @D5A
SIOCMSIFNETMASK   EQU    5013 Cmd for Set interface network
*                mask for an Internet address
*                (Pre-Router wrap) @D5A
SIOCMSIFMETRIC    EQU    5014 Cmd for Set Interface routing metric
*                (Pre-Router wrap) @D5A
SIOCMSRBRTTABLE   EQU    5015 Cmd for Set Routing table required
*                required request
*                (Pre-Router wrap) @D5A
SIOMSMETRIC1RT    EQU    5016 Cmd for Set metric1
*                (Pre-Router wrap) @D5A
SIOCMSICMPREDIRECT EQU    5017 Cmd for Propagating ICMP redirects
*                (Pre-Router wrap) @D5A
SIOCSETTKN        EQU    X'8008139A' 5018 Set Tcp/Ip master Tkn @P6A
*
SIOCMSADDRT6      EQU    X'8044F604' Add IPV6 Route @DDA
*
SIOCMSDELRT6      EQU    X'8044F605' Delete IPV6 Route @DDA
*
SIOCGRT6TABLE     EQU    X'C014F606' Get IPV6 Network Routing
*                Table @DDA
SIOCGRT6TABLE64   EQU    X'C018F606' Get IPV6 Network Routing
*                Table 64-BIT @PFC
SIOCMSRBRT6TABLE  EQU    X'8000F607' Rebuild IPV6 Route Tables @DDA
*
SIOCGHOMEIF6      EQU    X'C014F608' Get IPV6 Home Interface
*                Configuration @PBC
SIOCGHOMEIF664    EQU    X'C018F608' Get IPV6 Home Interface
*                Configuration 64-Bit @PFC
SIOCMSRBHOMEIF6   EQU    X'8000F609' Rebuild IPV6 Home Interface@PBC
*
SIOCMSCHGRT6METRIC EQU    X'8044F60A' Change IPV6 route's metric @PDA
*
SIOCMSMODHOMEIF6  EQU    X'8008F60B' Modify IPV6 Home Interface @DHA
*

```

```

SIOCMSADDRT6V2      EQU    X'8058F60C'  Add IPV6 route version 2    @PGA
*
SIOCMSDELRT6V2      EQU    X'8058F60D'  Del IPV6 route version 2    @PGA
*
SIOCMSCHGRT6METRICV2 EQU    X'8058F60E'  Change IPV6 route's metric @PGA
*
* Connection type and security credentials on TCPIP sockets.          @DRA
* Refer to: Comm Svr: IP Programmer's Guide and References.          @DRA
SIOCGPARTNERINFO     EQU    X'C000F612'  Get Info                    @DRA
SIOCSPARTNERINFO     EQU    X'8004F613'  Set Optimization            @DRA
* Ioctl Command Constants - terminal control
TIOCGWINSZ           EQU    X'4008A368'  get window size             @D3A
TIOCSWINSZ           EQU    X'8008A367'  set window size             @D3A
TIOCNOTIFY           EQU    X'8001A364'  notify master by packet     @P7A
* Constants for argument when TIOCNOTIFY is specified                @P7A
IOCC#PWBEGIN         EQU    1            Begin secure data         @P7A
IOCC#PWEND           EQU    2            End secure data          @P7A
*
* Ioctl command constants - for Router query                          @D5A
SIOCGRTTABLE         EQU    X'C008C980'  Gets Network Routing Tab    @D5A
SIOCGRTTABLE64       EQU    X'C00CC980'  Get NRT for 64-Bit C Pgm   @PCA
*
SIOCSETRTTD          EQU    X'8008C981'  Set Socket to be attached to
*                               1 TD                                         @D5A
*
SIOCMSMODHOMEIF      EQU    X'8008C983'  Modify Home Interface       @DHA
*
SIOCMSMODHOMEIFV2    EQU    X'8016C984'  Modify Home Interface V2    @PIA
*
SIOCMSADDRTV2        EQU    X'8054C985'  Cmd for Add Route V2       @PIA
*
SIOCMSDELRTV2        EQU    X'8054C986'  Cmd for Delete Route V2    @PIA
*
SIOMSMETRIC1RTV2     EQU    X'8054C987'  Cmd for Set Metric1 V2     @PIA
*
FIONBIO              EQU    X'8004A77E'  set/reset nonblock I/O
FIONREAD              EQU    X'4004A77F'  get number of readable bytes
*                               available
FIONWRITE             EQU    X'4004A78A'  get number of writeable bytes
*                               available
FIOASYNC              EQU    X'8004A77D'  set/clear async I/O        @D5A
FIOSETOWN             EQU    X'8004A77C'  set owner                   @D5A
FIOGETOWN             EQU    X'4004A77B'  get owner                   @D5A
SECIGET               EQU    X'4010E401'  get security information
SECIGET_T             EQU    X'4028E403'  Get peer task security      @DIA
SIOCTIEDESTHRD       EQU    X'8004E404'  Tie descriptor to thread    @DIA
SIOCSECEVNR          EQU    X'C012A78B'  SET/GET Client Security
*                               Environment                                  @DLA
SIOCADDRT             EQU    X'8030A70A'  IBM use only, Add routing
*                               table entry
SIOCATMARK            EQU    X'4004A707'  Is current location pointing
*                               to out-of-band data?
SIOCSPGRP             EQU    X'8004A708'  Set process group           @DDA
SIOCGPGRP            EQU    X'4004A709'  Get process group           @DDA
SIOCDELRT            EQU    X'8030A70B'  IBM use only, Delete routing
*                               table entry
SIOMETRIC1RT         EQU    X'8030A70C'  IBM use only, Set metric1 @D5A
SIOCSIFADDR          EQU    X'8020A70C'  Set Network interface addr@D5A
SIOCGIFADDR          EQU    X'C020A70D'  Get Network interface address
SIOCGIFBRDADDR       EQU    X'C020A712'  Get Network interface
*                               Broadcast Address
SIOCSIFBRDADDR       EQU    X'8020A713'  Sets Network interface
*                               Broadcast Address @D5A
SIOCGIFCONF          EQU    X'C008A714'  Get Network interface Config
SIOCGIFCONF64        EQU    X'C00CA714'  for 64-Bit C Pgms
SIOCGIFCONF6         EQU    X'C018A722'  Get IPv6 Network IfConf   @DJA
*
SIOCGIFMTU           EQU    X'C020A726'  Get MTU Size               @DQA
*
SIOCGIFDSTADDR       EQU    X'C020A70F'  Get Network interface
*                               Destination Address
SIOCGIFFLAGS         EQU    X'C020A711'  Get Network interface Flags
SIOCGIFMETRIC        EQU    X'C020A717'  IBM use only, Gets Network
*                               Interface Routing Metric
SIOCGIFNETMASK       EQU    X'C020A715'  Get Network interface
*                               Network Mask
SIOCSIFNETMASK       EQU    X'8020A716'  Set Network interface
*                               Network Mask @D5A
SIOCSIFDSTADDR       EQU    X'8020A70E'  IBM use only, Sets Network
*                               Interface Destination Address
SIOCSIFFLAGS         EQU    X'8020A710'  IBM use only, Sets Network
*                               Interface Flags

```

```

SIOCSIFMETRIC      EQU   X'8020A718'   IBM use only, Sets Network
*                                     Interface Routing Metric
SIOCSARP           EQU   X'8024A71E'   IBM use only, Sets ARP
*                                     Entry @D5A
SIOCGRP           EQU   X'C024A71F'   IBM use only, Gets ARP
*                                     Entry @D5A
SIOCGRP           EQU   X'8024A720'   IBM use only, Deletes ARP
*                                     Entry @D5A
SIOCSHIWAT        EQU   X'8004A700'   Set High Water Mark
*                                     (Not Supported) @D5A
SIOCGHIWAT        EQU   X'4004A701'   Get High Water Mark
*                                     (Not Supported) @D5A
SIOCSLOWAT        EQU   X'8004A702'   Set Low Water Mark
*                                     (Not Supported) @D5A
SIOCGLOWAT        EQU   X'4004A703'   Get Low Water Mark
*                                     (Not Supported) @D5A
FIOFCTLNBIO       EQU   X'0000E402'   change blocking/nonblocking
*
IOCC#ILINK         EQU   X'4004E21A'   I_LINK @D9A
*
IOCC#EDITACL      EQU   X'2000C100'   Edit ACL @P5A
*
SETFACL           EQU   X'0000D301'   SET FILE ACL @DBA
IOCC#SETFACL      EQU   X'0000D301'   SET FILE ACL @DCA
GETFACL           EQU   X'0000D302'   GET FILE ACL @DBA
IOCC#GETFACL      EQU   X'0000D302'   GET FILE ACL @DCA
*
*   Get Port of Entry for Multilevel Security @DDA
*   Get Port of Entry Attributes for a Socket Resource @DDA
SIOCGSOCKPOEATTRS EQU   X'4000D305'   @DDA
*   Get Port of Entry Attributes for a non-Socket Resource @DGA
SIOCGFDPOEATTRS  EQU   X'4000D306'   @DDA
*   Get Multilevel Security info for a Socket Resource @DKA
SIOCGSOCKMLSINFO EQU   X'4000D307'   @DKA
*   Constants for argument when FIONBIO is specified
IOCC#BLOCK       EQU   X'00000000'   Allow blocking to occur
IOCC#NONBLOCK    EQU   X'00000001'   Do not allow blocking to occur
*   Constants for argument when SIOCTIESTHRD is specified @DIA
SIOC#TIESD       EQU   X'00000001'   Tie descriptor to thread @DIA
SIOC#UNTIESD     EQU   X'00000000'   UnTie descriptor from thrd@DIA
*   Constants for argument when SIOCSECEVNR is specified @DLA
SIOC#SETENVNR    EQU   X'00000001'   SET Security Environment @DLA
SIOC#GETENVNR    EQU   X'00000002'   GET Security Environment @DLA
*
*****
*
*               I P v 6 @DDA *
*
*****
*   IPv6 Ioctl's
SIOCGIFVERSION   EQU   X'4000F601'   Get Interface Ver   Out
SIOCGSRCIPADDR   EQU   X'C000F602'   Get Source Addr   InOut
SIOCGIFNAMEINDEX EQU   X'4000F603'   Get If Name/Index   Out
*
*****
*
*   Get and Set ip_msfilter @DMA
*
*****
*   Get and Set ip_msfilter (IPv4 only) @DMA
SIOCGIPMSFILTER  EQU   X'C000A724'   @DMA
SIOCSIPMSFILTER  EQU   X'8000A725'   @DMA
*
*   Get and Set group_filter @DMA
*
*****
*   Get and Set group_filter (IPv6 or IPv4) @DMA
SIOCGMSFILTER    EQU   X'C000F610'   @DMA
SIOCSMSFILTER    EQU   X'8000F611'   @DMA
*
*****
*   Packet mode or Extended Packet mode data record control data. @D7C*
*
*   Returned on master read when no control information is pending. *
*   In packet mode one byte is returned. In extended packet mode, four *
*   bytes are returned. Data follows the control data. *
*****
TIOC_DATA        EQU   X'00'         Data packet @D3A
*****
*   Packet mode control byte - returned on master read() @D7C*
*

```

```

* A single control byte is returned in packet mode. In extended
* packet mode, four bytes are returned, with the non-extended bits
* in the fourth byte. The equates below can be used against the
* fourth byte (with TM, OI and NI) or against all four bytes (with
* OC, NC, etc.).
*****
TIOCPKT_FLUSHREAD EQU X'01'      Input was flushed      @D3A
TIOCPKT_FLUSHWRITE EQU X'02'     Output was flushed      @D3A
TIOCPKT_STOP EQU X'04'           Stop output             @D3A
TIOCPKT_START EQU X'08'          Start output            @D3A
TIOCPKT_NOSTOP EQU X'10'         STOP/START not standard @D3A
TIOCPKT_DOSTOP EQU X'20'         STOP/START standard     @D3A
*****
* Extended Packet mode control byte - returned on master read() @D7C*
*****
TIOCPKT_PASSTHRU EQU X'00000100' 3270 Passthrough mode @D7C
TIOCPKT_NOPASSTHRU EQU X'00000200' Not 3270 Passthrough mode @D7C
TIOCPKT_ECHO EQU X'00000400'      ECHO set on           @D7A
TIOCPKT_NOECHO EQU X'00000800'    ECHO set off          @D7A
TIOCPKT_CHCP EQU X'00001000'      Code page change       @D7A
TIOCPKT_PWBEGIN EQU X'00002000'    Begin secure data     @P7A
TIOCPKT_PWEND EQU X'00004000'     End secure data        @P7A
*****
* Get Pathname @DOA
IOCC#GETPATHNAME EQU 17           Absolute name        @DOA
IOCC#GETPATHNAMEREL EQU 19        Relative name         @DOA
*****
* UPDPTOFTE @D8C
*****
IOCC#UPDPTOFTE EQU 20             UPDATE OFTE CMD       @D8A
* @DDA
      AIF ('&DSECT' EQ 'NO').B411 @DDA
IOCUOFTE DSECT , ARGUMENT BUFFER @DDA
      AGO .C411 @DDA
      ANOP , @DDA
      .B411 @DDA
IOCUOFTE DS 0F ARGUMENT BUFFER @DDA
      .C411 ANOP , @DDA
* @DDA
IOCUOCMD DS F SUBCMD @D8A
IOCUO#READ EQU 1 READ @D8A
IOCUO#WRITE EQU 2 WRITE @D8A
IOCUO#CS EQU 3 COMPARE & SWAP @D8A
IOCUOVALUEBUFF DS 0F VALUE TO/FROM STATE AREA @D8A
IOCUOVFFSET DS F OFFSET (>=0) @D8A
IOCUOVLEN DS F LENGTH (>0) @D8A
IOCUOVDATA DS 0C DATA @D8A
* @DDA
      AIF ('&DSECT' EQ 'NO').B412 @DDA
IOCUOCSBUFF DSECT , COMPARE VALUE FOR CS SUBCMD @DDA
      AGO .C412 @DDA
      ANOP , @DDA
      .B412 @DDA
IOCUOCSBUFF DS 0F COMPARE VALUE FOR CS SUBCMD @DDA
      .C412 ANOP , @DDA
* @DDA
IOCUOCSOFFSET DS CL4 OFFSET (BYTE BDY) @D8A
IOCUOCSLEN DS CL4 LENGTH (BYTE BDY) @D8A
IOCUOCSDATA DS 0C DATA @D8A
*
IOCC#REGFILEINT EQU 21 REGISTER FILE INTR @DAA
IOCC#FASTPATH EQU 22 Set FastPath Ops @P9A
*
IOCC#DEVCONSOLE EQU 23 /dev/console behavior @DEA
IOCC#DEVCONSUPPRESS EQU 1 /dev/console - set suppress @DEA
IOCC#DEVCONUNSUPPRS EQU 0 /dev/console - unsuppress @DEA
*
IOCC#DEVFD EQU 27 /dev/fd behavior @DFA
IOCC#XATTR EQU 28 get/list xattr @DSA
* LFS/Cinet Level Ioctl @DDA
IOCC#GETSTACKS EQU 24 Get Stack Names @DDA
IOCC#DIRIOCTL EQU 25 Directed Ioctl @DDA
IOCC#GRTRSELECT EQU 26 Get PreRtr Select @DDA
*
*****
* @DDA*
* Ioctl#GetStacks -
* Get the names of the stacks that are attached to a socket.
*
*****
*
      AIF ('&DSECT' EQ 'NO').B413 @DDA
IOCSTACKINFO DSECT , @DDA
      AGO .C413 @DDA

```

```

.B413      ANOP      ,                                @DDA
IOCSTACKINFO      DS      0F                                @DDA
.C413      ANOP      ,                                @DDA
IOCSTACKINFOHEADER      DS      CL8
                                ORG      IOCSTACKINFOHEADER
IOCSTACKINFOFLAGS      DS      X      Flags
IOCSTACKCINET      EQU      X'80'      Cinet socket
                                DS      CL3
IOCSTACKENTRIES      DS      F      Number of Names returned
                                ORG
IOCSTACKNAMES      DS      CL16      Array of stack names
*****
*      Array of IOCSTACKNAMES      *
*****
*
      AIF      ('&DSECT' EQ 'NO').B414                                @DDA
IOCSTACKNAMESD      DSECT      ,                                @DDA
      AGO      .C414                                @DDA
.B414      ANOP      ,                                @DDA
IOCSTACKNAMESD      DS      0F                                @DDA
.C414      ANOP      ,                                @DDA
IOCSTACKNAME      DS      CL8      Stack name
IOCSTACKTDINDEX      DS      X      Cinet Stack TdIndex
IOCSTACKFLAGS      DS      X      Flags
IOCSTACK_ACTIVE      EQU      X'80'      Active
IOCSTACK_IPV6_SUPPORT      EQU      X'40'      IPv6 is supported
IOCSTACK_IPV6_INTERFACES      EQU      X'20'      IPv6 Home Interfaces
IOCSTACK_IPV4_INTERFACES      EQU      X'10'      IPv4 Home Interfaces
                                DS      CL6

*****
*                                @DDA*
*      Ioccc#DirIoctl - Directed Ioctl      *
*      Passes the imbedded ioctl to the specified stack.      *
*                                *
*****
*
      AIF      ('&DSECT' EQ 'NO').B415                                @DDA
IOCDDIRIOCTL      DSECT      ,                                @DDA
      AGO      .C415                                @DDA
.B415      ANOP      ,                                @DDA
IOCDDIRIOCTL      DS      0F                                @DDA
.C415      ANOP      ,                                @DDA
IOCDDIRHDR      DS      CL16
                                ORG      IOCDDIRHDR
IOCDDIRNAME      DS      CL8      Target Stack Name
IOCDDIRCMD      DS      XL4      Imbedded ioctl Command
IOCDDIRARGLEN      DS      F      Imbedded ioctl Argument Length
                                ORG
IOCDDIRARG      DS      C      Imbedded ioctl Argument

*****
*                                @DDA*
*      Ioccc#GRtrSelect - Get Cinet PreRouter's selected stack for each *
*      of an array of specified destination IP addresses.      *
*                                *
*****
*
      AIF      ('&DSECT' EQ 'NO').B416                                @DDA
IOCRTRSELECT      DSECT      ,                                @DDA
      AGO      .C416                                @DDA
.B416      ANOP      ,                                @DDA
IOCRTRSELECT      DS      0F                                @DDA
.C416      ANOP      ,                                @DDA
IOCRTRIPADDR      DS      CL16      Input IP Address
IOCRTRSTACK      DS      CL8      Output Selected Stack Name
                                ORG      IOCRTRSTACK
IOCRTRERRTEST      DS      CL1      Error if = 0
                                DS      CL1
IOCRTRERRNO      DS      XL2      Error RC (Errno)
IOCRTRRSN      DS      XL4      Error Rsn (ErrnoJr)
                                ORG

IOCRTRERROR      EQU      X'00'      IocRtrErrTest value to test for error

*****
*                                @DDA*
*      SiocGIfNameIndex - Get Interface Name/Index Table      *
*                                *
*****
*
      AIF      ('&DSECT' EQ 'NO').B418                                @DDA

```

```

IF_NAMEINDEXENTRY DSECT , @DDA
    AGO .C418 @DDA
.B418 ANOP , @DDA
IF_NAMEINDEXENTRY DS 0F @DDA
.C418 ANOP , @DDA
IF_NIINDEX DS F Interface Index
    ORG IF_NIINDEX
IF_NITDINDEX DS H CInet Td Index
IF_NIIFINDEX DS H Stack Interface Index
    ORG
IF_NINAME DS CL16 Interface Name, blank padded
IF_NIEXT DS CL4
    ORG IF_NIEXT
IF_NINAMETERM DS CL1 Null for C for Name len=16
IF_NIFLAGS DS X Name Index Flags @PKA
IF_NIOSM EQU X'80' 1 = OSM Interface @PKA
    DS CL2 Reserved @PKC
    ORG
IF_NAMEINDEXENTRYL EQU *-IF_NAMEINDEXENTRY @DDA

    AIF ('&DSECT' EQ 'NO').B417 @DDA
IF_NAMEINDEX DSECT , @DDA
    AGO .C417 @DDA
.B417 ANOP , @DDA
IF_NAMEINDEX DS 0F @DDA
.C417 ANOP , @DDA
IF_NIHEADER DS 2F
    ORG IF_NIHEADER
IF_NITOTALIF DS F Total Active Interfaces on System
IF_NIENTRIES DS F Number of entries returned
    ORG
IF_NITABLE DS CL(IF_NAMEINDEXENTRYL)

*****
* @DDA *
* SiocGSockPoeAttr - Socket Port of Entry Attributes *
* *
*****
    AIF ('&DSECT' EQ 'NO').B419 @DDA
IOCP0EATTR DSECT , @DDA
    AGO .C419 @DDA
.B419 ANOP , @DDA
IOCP0EATTR DS 0F @DDA
.C419 ANOP , @DDA
IOCP0EPEERIPADDR DS CL16 Peer IP Address
    ORG IOCP0EPEERIPADDR
IOCP0EPEERIPV6PREFIX DS CL12
IOCP0EPEERIPV4ADDR DS F
    ORG
IOCP0ETERMID DS CL8 TERMINAL Profile Name
IOCP0ELABEL DS CL8 Security Label
IOCP0EPROFILE DS CL64 SERVAUTH Resource Name

*****
* @DKA *
* SiocGSockMLSINFO - Socket Multilevel Security Information *
* IocPoeProfile returns full resource name *
* IocMlsProfile returns actual profile name in use *
* *
*****
    AIF ('&DSECT' EQ 'NO').B420 @DKA
IOCMLSINFO DSECT , @DKA
    AGO .C420 @DKA
.B420 ANOP , @DKA
IOCMLSINFO DS 0F @DKA
.C420 ANOP , @DKA
IOCMLSMVSNAME DS CL8 MVS System Name @DKA
IOCMLSSTKNAME DS CL8 Stack Job Name @DKA
IOCMLSNAZNAME DS CL8 NetAccess Zone Name @DKA
IOCMLSUSRNAME DS CL8 Caller UserID @DKA
IOCMLSUSRLBL DS CL8 Caller Security Label @DKA
IOCMLSSTKLBL DS CL8 Stack Security Label @DKA
IOCMLSNAZLBL DS CL8 Zone Security Label @DKA
IOCMLSCONLBL DS CL8 Connct Security Label @DKA
IOCMLSPROFILE DS CL64 SERVAUTH Profile Name @DKA

*-----*
* Multicast Source Filter Structures from RFC 3678 *
* *
* *** These require the inclusion of BPXYSOCK *** *

```



```

*----- @DMA
*
*      AIF ('&MCAST' EQ 'NO').NOMCAST @DMA
*
*      SiocGIPMSFilter - Get a list of multicast source addresses @DMA
*      SiocSIPMSFilter - Set a list of multicast source addresses @DMA
*
*      AIF ('&DSECT' EQ 'NO').B421 @DMA
IP_MSFILTER DSECT , @DMA
      AGO .C421 @DMA
      .B421 ANOP , @DMA
IP_MSFILTER DS 0F @DMA
      .C421 ANOP , @DMA
IMSF_HEADER DS 0C Header @DMA
IMSF_MULTIADDR DS CL4 IP Multicast address of group @DMA
IMSF_INTERFACE DS CL4 Local IP addr of interface @DMA
IMSF_FMODE DS CL4 Filter mode @DMA
IMSF_NUMSRC DS CL4 Number of sources in src_list @DMA
IMSF_HEADER_LEN EQU *-IP_MSFILTER @DMA
IMSF_SLIST DS 0CL(L'IMSF_SRCADDR) Start of source list @DMA
*
*      AIF ('&DSECT' EQ 'NO').B422 @DMA
IMSF_SRCENTRY DSECT , @DMA
      AGO .C422 @DMA
      .B422 ANOP , @DMA
IMSF_SRCENTRY DS 0F Source list entry @DMA
      .C422 ANOP , @DMA
IMSF_SRCADDR DS CL4 Source IP address @DMA
IMSF_SRCENTRY_LEN EQU *-IMSF_SRCENTRY Length @PHA
*
* @DMA
* SiocGMSFilter - Get a list of multicast source addresses @DMA
* SiocSMSFilter - Set a list of multicast source addresses @DMA
*
*****
* GROUP_REQ STRUCTURE *
*
***** @DMA
*      AIF ('&DSECT' EQ 'NO').B423 @DMA
GROUP_FILTER DSECT , @DMA
      AGO .C423 @DMA
      .B423 ANOP , @DMA
GROUP_FILTER DS 0F @DMA
      .C423 ANOP , @DMA
GF_HEADER DS 0C Header @DMA
GF_INTERFACE DS CL4 Interface index @DMA
      DS CL4 Padding @DMA
GF_GROUP DS CL(L'SOCKADDR_STORAGE) Group address @DMA
GF_FMODE DS CL4 Filter mode @PHM
GF_NUMSRC DS CL4 Number of sources @PHM
GF_HEADER_LEN EQU *-GROUP_FILTER @PHM
GF_SLIST DS 0CL(L'GF_SRCENTRY) Start of source list @DMA
      ORG GF_GROUP @DMA
GF_MULTISOCKADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN) @DMA
      ORG GF_GROUP @DMA
GF_MULTISOCKADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN) @DMA
*
*      AIF ('&DSECT' EQ 'NO').B424 @DMA
GF_SRCENTRY DSECT , @DMA
      AGO .C424 @DMA
      .B424 ANOP , @DMA
GF_SRCENTRY DS 0F Source list entry @DMA
      .C424 ANOP , @DMA
GF_SRCADDR DS CL(L'SOCKADDR_STORAGE) Source address @DMA
GF_SRCENTRY_LEN EQU *-GF_SRCENTRY Length @PHA
      ORG GF_SRCADDR @DMA
GF_SRCADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN) @DMA
      ORG GF_SRCADDR @DMA
GF_SRCADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN) @DMA
*
MCAST_INCLUDE EQU 0 @DMA
MCAST_EXCLUDE EQU 1 @DMA
MCAST_NUMSRC_MAX EQU 64 Max number of sources for @PHA
*      GF_NUMSRC and IMSF_NUMSRC @DPA
* @DMA
* .NOMCAST ANOP , End of Multicast Structures @DMA
*****
* SIOCGRRTABLE - Obtain route information. Returns information for *
*      IPv4 routes from the TCP/IP stack's main route table.*
*
*
*      The Route entry structures returned can either be *

```

```

*          Version 1, Version 2, or Version 3 structures.
*          Version 1
*          - Field IOCN_IPADDRRTMSGHOMEIF contains an IP
*            address
*          Version 2
*          - IOCN_RTMSGTYPE structure is the same size
*            as Version 1
*          - Field IOCN_IPADDRRTMSGHOMEIF contains an
*            interface index.
*          Version 3
*          - IOCN_RTMSGTYPE structure is larger than the
*            Version 1 or 2 size
*          - Field IOCN_IPADDRRTMSGHOMEIF contains an
*            interface index
*          - New MTU field added
*          The RTEV3 macro variable controls whether the
*          Version 3 Route entry structure is generated by
*          this macro. By default, the macro generates the
*          Version 3 Route entry structure.
*
*          When requesting Version 2 or Version 3 output in a
*          CINET environment, invoking applications must either
*          have stack affinity or use the IOCC#DIRIOCTL to
*          invoke this ioctl. Otherwise, the output from the
*          first stack will be in the requested version format
*          but, the output from subsequent stacks will be in
*          Version 1 format.
*
* Input    - Input to the ioctl is a buffer length and a buffer
*            address. By default, Version 1 Route entry structures
*            are returned.
*
*            To obtain Version 2 or 3 Route entry output, an
*            IOCN_RTMSGHDRTYPE structure must be setup at the
*            beginning of the output buffer, before invoking the
*            ioctl. The following fields must be set:
*            - IOCN_RTMSGHDRVVER set to Version 2 or 3
*            - IOCN_RTMSGHDREYEID set to the correct eyecatcher
*              value.
*
* Output    - Return_value = 0
*            - IOCN_RTMSGHDRTYPE structure returned with
*              field IOCN_RTMSGHDRNUMENT set to the number
*              of Route entry structures returned. In the
*              output buffer, the header is followed by the
*              Route entry structures.
*            - The IOCN_RTMSGHDRVVER and IOCN_RTMSGHDRSUBVER
*              fields indicate whether Version 1, 2, or 3
*              of the Route entry structures was returned.
*            - IOCN_RTMSGHDRNUMENT - number of either Version
*              1, 2, or 3 Route entry structures returned.
*            - Return_value = -1
*            - Return_code = EBADMSG
*              This means that the output buffer is not large
*              enough for one IOCN_RTMSGHDRTYPE structure
*              and one Route entry structure.
*            - Return_code = ERANGE
*              This means that the output buffer is not large
*              enough for all the Route entry structures which
*              can be returned. In this case, only the
*              IOCN_RTMSGHDRTYPE structure will be returned.
*              Field IOCN_RTMSGHDRNUMENT in this header will
*              contain the number of Route entry structures
*              which can be returned. Field IOCN_RTMSGHDRVVER
*              will indicate which version of the Route entry
*              structures will be returned.
*
*          @PIA
*          *****
*          * Header structure - IOCN_RTMSGHDRTYPE
*          *
*          *          AIF ('&DSECT' EQ 'NO').B425
*          *          IOCN_RTMSGHDRTYPE DSECT , Route header
*          *          AGO .C425
*          *          .B425 ANOP ,
*          *          IOCN_RTMSGHDRTYPE DS 0F Route header
*          *          .C425 ANOP ,
*          *
*          *          IOCN_RTMSGHDRLEN DS H Length of header
*          *          IOCN_RTMSGHDRVVER DS FL1 Version of Route entry structures
*          *          returned
*          *
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA
*          @PIA

```

```

DS      CL1  Reserved                                @PIA
IOCN_RTMSGHREYEID DS      CL4  Eye catcher 'Rmsg'        @PIA
IOCN_RTMSGHREYEC EQU      C'Rmsg' Eye catcher constant  @PIA
IOCN_RTMSGHDRSUBVER DS      FL1  Version of Route entry structures
*                                     returned          @PIA
DS      CL3  Reserved                                @PIA
IOCN_RTMSGHDIOTLNUM DS      F    Set to x'10' to indicate the
*                                     SIOCGRTTABLE ioctl  @PIA
IOCN_RTMSGHDRNUMENT DS      F    Output - number of Route entries
*                                     returned          @PIA
DS      CL52 Reserved                                @PIA
IOCN_RTMSGHDR#LEN EQU      *-IOCN_RTMSGHDRTYPE Length of header@PIA
*
* Version constants for fields IOCN_RTMSGHDRVER and
* IOCN_RTMSGHDRSUBVER                                @PIA
*
IOCN_RTMSGTYPEV1 EQU      1      Version 1              @PIA
IOCN_RTMSGTYPEV2 EQU      2      Version 2              @PIA
IOCN_RTMSGTYPEV3 EQU      3      Version 3              @PIA
*
* Version 1 and Version 2 Route entry structure - IOCN_RTMSGTYPE @PIA
*
      AIF ('&DSECT' EQ 'NO').B426                      @PIA
IOCN_RTMSGTYPE DSECT ,      Route entry structure      @PIA
      AGO .C426                                         @PIA
.B426 ANOP ,                                           @PIA
IOCN_RTMSGTYPE DS      0F      Route entry structure  @PIA
.C426 ANOP ,                                           @PIA
*
IOCN_IPADDRRTMSGDEST DS      CL4  Destination IP address @PIA
IOCN_IPADDRRTMSGMASK DS      CL4  Subnet mask           @PIA
IOCN_IPADDRRTMSGGATE DS      CL4  Gateway IP address    @PIA
IOCN_IPADDRRTMSGHOMEIF DS      F    Version 1 = IP address @PIA
*                                     Other versions = interface
*                                     index              @PIA
IOCN_RTMSGMETRICTYPE DS      F    Metric type is always 1, meaning
*                                     metric value is in hop counts @PIA
IOCN_RTMSGMETRIC DS      F    Metric value in hop counts @PIA
DS      CL3  Reserved                                @PIA
IOCN_RTATTRRTMSG DS      XL1  Route flags              @PIA
IOCN_BRTATTRLOOPBACK EQU      X'80' 1 = Loopback interface @PIA
IOCN_BRTATTRLOCAL EQU      X'40' 1 = Local/Home IP address @PIA
IOCN_BRTATTRDYNBUILT EQU      X'10' 1 = Dynamically built, e.g.
*                                     by ICMP redirect   @PIA
IOCN_BRTATTRHOST EQU      X'04' 1 = Host route, 0 = Network
*                                     route              @PIA
IOCN_BRTATTRGATEWAY EQU      X'02' 1 = Gateway          @PIA
IOCN_BRTATTRRTUP EQU      X'01' 1 = Route is active, 0 = Route
*                                     is inactive        @PIA
IOCN_RTMSGRTETYPE DS      XL1  Route type              @PIA
IOCN_RTOTHER EQU      1      Other (default, direct)    @PIA
IOCN_RTLOCAL EQU      2      Static (configured)        @PIA
IOCN_RTICMP EQU      4      ICMP                      @PIA
IOCN_RTRIP EQU      8      RIP                        @PIA
IOCN_RTOSPF EQU      13     OSPF                      @PIA
IOCN_RTRESTAT EQU      130   Replaceable static        @PIA
DS      CL3  Reserved                                @PIA
IOCN_RTMSGV1#LEN EQU      *-IOCN_RTMSGTYPE V1 Route entry len @PIA
IOCN_RTMSGV2#LEN EQU      *-IOCN_RTMSGTYPE V2 Route entry len @PIA
      AIF ('&RTEV3' EQ 'NO').B427                      @PIA
*
* Version 3 Route entry structure - additional field @PIA
*
IOCN_RTMSGMTU DS      H      Route's MTU value          @PIA
DS      H      Reserved                                @PIA
DS      F      Reserved                                @PIA
DS      F      Reserved                                @PIA
DS      F      Reserved                                @PIA
IOCN_RTMSGV3#LEN EQU      *-IOCN_RTMSGTYPE V3 Route entry len @PIA
.B427 ANOP ,                                           @PIA
*
*****
* IOCN_IFREQ @DQA*
* Mapping that defines the network interface block that is used *
* on ioctls that transfer network interface information *
* *
* This is equivalent to the C ifreq structure from in.h *
* This is the ASM version of the PL/X IOCN_IfType from BPXZIOCN. *
* *
*****
*
      AIF ('&DSECT' EQ 'NO').B428

```

```

IOCN_IFREQ DSECT      Mapping for network interface information
      AGO      .C428
      .B428     ANOP      ,
IOCN_IFREQ DS 0F      Mapping for network interface information
      .C428     ANOP      ,
IOCN_IFTYPE DS 0F      Structure Name from BPXZIOCN.
IOCN_IFNAME DS CL16     Interface name.
IOCN_IFUNION DS 0CL16   Union of fields:
      ORG      IOCN_IFUNION
IOCN_SADDRIF DS CL16    The address of the interface
      ORG      IOCN_IFUNION
IOCN_SADDRIFDEST DS CL16 Destination address in a point to point link
      ORG      IOCN_IFUNION
IOCN_SADDRIFBROADCAST DS CL16 Address for Broadcasting
      ORG      IOCN_IFUNION
IOCN_IFMETRIC DS F      Interface metric
      ORG      IOCN_IFUNION
IOCN_PIFDATA DS A      Pointer to an area set by TCPIP
      ORG      IOCN_IFUNION
IOCN_MTUSIZE DS F      MTU size. Used with SIOCGIFMTU.
      ORG      IOCN_IFUNION
IOCN_IFATTRIF DS 0BL2   Flag area.
*
IOCN_IFATTRBYTE1 DS 0CL1
IOCN_BIFATTRSNAP EQU X'20'
IOCN_BIFATTRTOKBRIDGE EQU X'10'
IOCN_BIFATTRCHECKSUM EQU X'04'
IOCN_BIFATTRALLMULTI EQU X'02'
IOCN_BIFATTRALLPACKSUPT EQU X'01'
      ORG      IOCN_IFATTRBYTE1+1
IOCN_IFATTRBYTE2 DS 0CL1
IOCN_BIFATTRARPNOTSUPT EQU X'80'
IOCN_BIFATTRRESALLOC EQU X'40'
IOCN_BIFATTRNOTTRAILER EQU X'20'
IOCN_BIFATTRPTTOPT EQU X'10'
IOCN_BIFATTRLOOPBACK EQU X'08'
IOCN_BIFATTRRDEBUG EQU X'04'
IOCN_BIFATTRBROADCAST EQU X'02'
IOCN_BIFATTRUP EQU X'01'
*
      ORG      IOCN_IFUNION+16
IOCN_IFEND DS 0C      End of structure
IOCN_IFREQ_LEN EQU *-IOCN_IFREQ Length of Structure
IOCN_#IFNAMESIZE EQU 16 Size of the name field
*
** BPXYIOCC End

```

## BPXYIOC6 – Map IPV6 prerouter structures

BPXYIOC6 is used by transport providers. DSECT= is allowed but ignored. AMODE 64 callers use “BPXYIOC6 – Map IPV6 prerouter structures” on page 1196.

```

      BPXYIOC6      ,
NETCONFHDR      DSECT ,
* ----- 32-Bit Version @P2A
NCHEYECATCHER DS CL4 Eye catcher
NCHIOCTL DS F Ioctl being processed (RAS)
NCHBUFFERLENGTH DS F Buffer Length
NCHBUFFERPTR DS F Buffer Pointer
NCHNUMENTRYRET DS F Number of HomeIf returned via *
      SIOCGHOMEIF6 or the number of *
      GRT6RtEntry's returned via *
      SIOCGRT6TABLE.
NETCONFHDR#LENGTH EQU *-NETCONFHDR Length of NETCONFHDR
*
*****
* HomeIf Structure *
*****
*
HOMEIF DSECT , HomeIf structure
HomeIfAddress DS CL16 Home Interface Address
*
HomeIf#LENGTH EQU *-HOMEIF Length of HOMEIF
*
*****
* GRT6RtEntry Structure *
*****

```

```

*
GRT6RTENTRY      DSECT ,      GRT6RtEntry Structure
*
GRT6DESTINATION  DS      CL16 Destination IP Address
GRT6GATEWAY      DS      CL16 First HOP on the trip if going through *
                        a gateway
GRT6DESTPREFIXLEN DS      F      Destination's Prefix Length which is a *
                        decimal value that specifies how many *
                        of the leftmost contiguous bits of the*
                        address comprise the prefix
GRT6RTMETRIC      DS      F      Metric - hop count. Currently Tcp/Ip *
                        returns 1 for indirect routes and 0 *
                        for direct routes. If route is from *
                        routing daemon, metric is whatever *
                        routing daemon set it to.
GRT6RTFLAGS      DS      F      IPV6 Route Flags. Mapped by
*                               IPV6RtFlags structure @P5C
*
GRT6RTENTRY#LENGTH EQU  *-GRT6RTENTRY Length of GRT6RTENTRY
*
*****
* RT6Entry Structure *
*****
*
RT6ENTRY          DSECT ,      RtEntry Structure
*
RT6DESTINATION    DS      CL28 Destination IP address (in an IPV6 *
                        sockaddr structure)
RT6GATEWAY        DS      CL28 First HOP on the trip if going *
                        through a gateway (in an IPV6 *
                        sockaddr structure)
RT6DESTPREFIXLEN  DS      F      Destination's Prefix Length, *
                        which is a decimal value *
                        that specifies how many of *
                        the leftmost contiguous *
                        bits of the address *
                        comprise the prefix.
RT6METRIC          DS      F      Metric - hop count *
                        Currently Tcp/IP returns *
                        1 for indirect route and *
                        0 for direct route. *
                        If route is from routing *
                        daemon, metric is whatever *
                        routing daemon set it to.
RT6FLAGS          DS      F      IPV6 Route Flags.
*
RT6ENTRY#LENGTH  EQU  *-RT6ENTRY Length of RT6ENTRY
*
*****
* GRT6RtEntryV2 Structure *
*****
*
GRT6RTENTRYV2     DSECT ,      New Route Entry used with DCR A846 - *
                        Route Modification
*
GRT6OLDRTENTRY    DS      CL44 Old GRT6 Route Entry
GRT6RTHOMEIFIDX   DS      F      Route's Home Interface Idx
GRT6RTIFINDEX     DS      F      Route's Interface Index @P5A
GRT6RTMTU         DS      H      Route's MTU Value @P5A
                  DS      H      Reserved @P5C
                  DS      F      Reserved @P5C
                  DS      F      Reserved @P5C
*
GRT6RTENTRYV2#LENGTH EQU  *-GRT6RTENTRYV2 Length of GRT6RTENTRYV2
*
*****
* RT6EntryV2 Structure *
*****
*
RT6ENTRYV2        DSECT ,      New Route Entry Used with A846 *
                        MSADDRT6V2/MSDELRT6V2 IOCTLs
*
RT6OLDENTRY       DS      CL68 Old Route Entry used before A846 *
                        with SIOCMSADDRT6/SIOCMSDELRT6 IOCTL
RT6RTHOMEIFIDX    DS      F      Route's Home Interface Idx
                  DS      F      Reserved
                  DS      F      Reserved
                  DS      F      Reserved
                  DS      F      Reserved
*
RT6ENTRYV2#LENGTH EQU  *-RT6ENTRYV2 Length of RT6ENTRYV2
*

```

```

*****
* IPv6RtFlags Structure
*****
*
* IPv6RTFLAGS          DSECT ,          IPv6RtFlags Structure
*
* IPv6FLGROUTETYPE     DS    XL1    Route Type                @D1C
* IPv6FLGBYTE2         DS    XL1    Reserved
* IPv6FLGBYTE3         DS    XL1    Reserved
* IPv6FLGBYTE4         DS    XL1    FLAGS:
*                        EQU    X'80'  Reserved
*                        EQU    X'40'  Reserved
* IPv6BITV3            EQU    X'20'  1 = Version 3 fields included: @P5A
*                                - interface index
*                                - MTU value
* IPv6BITLOOPBACK      EQU    X'10'  1 = Loopback Interface
* IPv6BITHOME          EQU    X'08'  1 = Home interface
* IPv6BITHOST          EQU    X'04'  1 = Host Route. 0 = Network Route
* IPv6BITGATE          EQU    X'02'  1 = Gateway
* IPv6BITRTUP          EQU    X'01'  1 = Route is active
*
* *-----*
* * SiocGifConf6 - Get IPv6 Interface Configuration.                @D3A*
* *
* * Net_IfConf6Header is passed as the argument of the ioctl and    *
* * is returned with the number of entries and entry length of the  *
* * Net_IfConf6Entry structs that were written to the output buffer.*
* *
* * If Buflen=0=Buffer a Query function is performed and the        *
* * header is returned with: (1) the maximum supported version,    *
* * (2) the total number of entries that would be output and        *
* * (3) the length of each individual entry.                        *
* *
* * If a call to get information fails with RC=ERANGE or with        *
* * (RC=EINVAL & Nif6h_Version is changed) the call is converted  *
* * into a Query function and the content of the output buffer      *
* * is unpredictable.                                               *
* *
* * For information on the data returned in this structure refer    *
* * to the z/OS Communication Server's IP Configuration Guide and  *
* * IPv6 Network and Application Design Guide.                     *
* *
* *-----*
* NET_IFCONF6HEADER DSECT Header                                @D3A
* NIF6H_VERSION DS F      Input for Get IfConf6 Output for Query
* NIF6H_ENTRIES DS F      Output: number of entries returned in output *
*                        buffer
* NIF6H_ENTRYLEN DS F     Output: length of an entry
* NIF6H_BUFLen DS F       Input: length of buffer
* NIF6H_BUFFER64 DS 0CL8  Input: Amode(64) Buffer ptr
* NIF6H_BUFFER64H DS F
* NIF6H_BUFFER DS A       Input: Amode(31) Buffer ptr to output buffer *
*                        that will be filled with an array of      *
*                        Net_IfConf6Entrys.                        *
* NET_IFCONF6HEADER_LEN EQU *-NET_IFCONF6HEADER
*
* NET_IFCONF6ENTRY DSECT Entry                                @D3A
* NIF6E_NAME DS CL16     x00 interface name (blank padded - no null)
* NIF6E_STACKNAME DS CL8  x10 tcpip stack name (blank padded - no null)
* NIF6E_ADDR DS CL28      x18 Sock_Inet6_SockAddr of the interface
* NIF6E_ROUTEMETRIC DS F  x34 route metric
* NIF6E_PREFIXLEN DS H    x38 routing prefix length
* NIF6E_PREFIXORIGIN DS X x3A prefix origin, see below
* NIF6E_STATUS DS X       x3B status, see below
* NIF6E_FLAGS DS 0BL4     x3C Flags:
* NIF6E_FLAGS1 DS B
* NIF6E_FLAGS2 DS B
* NIF6E_FLAGS3 DS 0B
* NIF6E_VIRTUAL EQU X'40'
* NIF6E_MULTIPPOINT EQU X'08'
* NIF6E_MULTICASTCAPABLE EQU X'04'
* ORG NIF6E_FLAGS3+1
* NIF6E_FLAGS4 DS 0B
* NIF6E_POINT2POINT EQU X'10'
* NIF6E_LOOPBACK EQU X'08'
* NIF6E_ONLINK EQU X'01'
* ORG NIF6E_FLAGS+4
* NIF6E_MTU DS F          x40 mtu
*
* *****
* *
* * Constants for nif6h_version                                @D3A

```

```

*      *
*      *****
*
*
*
NIF6H#VER EQU 1          Current Version
NIF6H#VER1 EQU 1         Initial Version
*
*      *****
*      *
*      * Constants for nif6e_prefixorigin @D3A
*      *
*      *****
*
*
NIF6H#WELLKNOWN EQU 1
NIF6H#MANUAL EQU 2
NIF6H#RTRADV EQU 3
NIF6H#OTHER EQU 8
*
*      *****
*      *
*      * Constants for nif6e_status @D3A
*      *
*      *****
*
*
NIF6H#PREFERRED EQU 1
NIF6H#DEPRECATED EQU 2
NIF6H#INVALID EQU 3
NIF6H#INACCESSIBLE EQU 4
NIF6H#UNKNOWN EQU 5
NIF6H#TENTATIVE EQU 6
NIF6H#DUPLICATE EQU 7
NET_IFCONF6ENTRY_LEN EQU *-NET_IFCONF6ENTRY
*                               End SiocGifConf6 ----- @D3A
*
* *****
* *
* * Constants
* *
* *****
*
*
IOC6_#HOMEIFPREFIXLEN EQU 128 The prefix length for a home interface *
                               address returned on the SIOCGHOMEIF6 IOCTL.
IOC6_NCH#EYE EQU C'6NCH' IPV6 Network Configuration Header EyeCatcher.
IOC6_NCH64#EYE EQU C'6N64' IPV6 NetConfHdr EyeCatcher 64-BIT
*
* *****
* *
* * Maximum hop count for the Metric fields:
* *   GRT6RtMetric
* *   Rt6Metric
* *
* *****
*
*
IOC6_#MAXHOPMETRIC EQU 16
*
* *****
* *
* * Constants used for size of control areas
* *
* *****
*
*
IOC6_#MAXROUTES EQU 600
IOC6_#GRT6ROUTELEN EQU 44
*
* *****
*   Initial buffer size for SIOCGHOMEIF6 and SIOCGRT6TABLE.
*   *****
*
*
IOC6_#MAXGRT6LEN EQU 26400
IOC6_#NETCONFHDRLEN EQU 20
IOC6_#GRT6V2ROUTELEN EQU 64
IOC6_#MAXGRT6V2LEN EQU 38400
*
** BPXYIOC6 End

```

## BPXYIOV – Map the I/O vector structure

BPXYIOV is used by readv(), writev(), sendmsg() and recvmsg(). AMODE 64 callers use “BPXYIOV – Map the I/O vector structure” on page 1199.

```

                BPXYIOV
** BPXYIOV: Socket I/O Vectors
** Used By: FCT OPN
IOV              DSECT ,
IOV_ENTRY        DS      0F      Array Entry
* ----- 31-bit format
IOV_BASE         DS      A      Address of buffer
IOV_LEN          DS      F      Length of buffer
*
IOV#LENGTH       EQU      *-IOV_ENTRY Length of this structure
IOV_MAX          EQU      120     Maximum number of entries
** BPXYIOV End

```

## BPXYIPCP – Map interprocess communication permissions

```

                BPXYIPCP
** BPXYIPCP: Interprocess Communications Permission
** Used By: MCT, MGT, SCT, SGT, QCT, QGT
IPC_PERM         DSECT ,      Interprocess Communications
IPC_UID          DS      F      Owner's effective user ID
IPC_GID          DS      F      Owner's effective group ID
IPC_CUID         DS      F      Creator's effective user ID
IPC_CGID         DS      F      Creator's effective group ID
IPC_MODE         DS      XL4    Mode, mapped by BPXYMODE
IPC#LENGTH       EQU      *-IPC_PERM Length of Interprocess Control block
SPACE ,
* Key:
IPC_PRIVATE      EQU      0      Private key.
SPACE ,
* Mode bits:
IPC_CREAT        EQU      1      Map over S_TYPE in BPXYMODE
IPC_EXCL         EQU      2      Create entry if key does not exist.
IPC_MEGA         EQU      4      Fail if key exists.
IPC_BINSEM       EQU      4      Allocation in meg
IPC_RCVTYPEPID   EQU      4      Binary semaphore
IPC_SNDTYPEPID   EQU      4      Msgrcv TYPE=PID
IPC_PL01         EQU      8      Msgsnd TYPE=PID
IPC_PL01         EQU      16     Use PLO for serialization
IPC_SHORTHOLD    EQU      16     Binary semaphore short @D5A
IPC_PL02         EQU      32     Use PLO if practical
IPC_PLOINUSE     EQU      1      PLO is in use (_getipc only)
IPC_GIGA         EQU      8      Allocation in Gig - amode 64 @D6A
IPC_BELOWBAR     EQU      16     Allocate below bar @D6A
IPC_SHAREAS      EQU      32     Share within Address Space @D7A
SPACE ,
* Flag bits - semop, msgrcv, msgsnd:
IPC_NOWAIT       EQU      1      Error if request must wait.
SPACE ,
* Control Command:
IPC_RMID         EQU      1      Remove identifier.
IPC_SET          EQU      2      Set options.
IPC_STAT         EQU      3      Access status.
SPACE ,
* CONSTANTS WHICH MAP OVER BYTE S_TYPE, SEE BPXYMODE
** BPXYIPCP
End

```

## BPXYIPCQ – Map w\_getipc structure

AMODE 64 callers use “BPXYIPCQ – Map w\_getipc structure” on page 1199.

```

                BPXYIPCQ
*****
*
* BPXYIPCQ: w_getipc interface mapping
*

```



```

*   Used By: BPXGXGET
*
*****
IPCQ      DSECT ,      Interprocess Communications - Query
IPCQLENGTH DS   F      IPCQ#LENGTH used by system call.  If not
*                                     equal, check BPXYIPCQ and system levels.
IPCQTYPE   DS   CL4     "IMSG", "ISEM", "ISHM", "OVER", "IMAP"
IPCQOVER   DS   0D      OVERVIEW MAPPING STARTS HERE
*-----*
*   For IPCQTYPE = OVER, data starts here and the rest of the fields
*   in this section of code are not filled in.
*-----*
IPCQMID    DS   FL4     MEMBER ID
IPCQKEY    DS   XL4     KEY
IPCQIPCP   DS   CL20    MAPPED BY BPXYIPCP
IPCQGTIME  DS   XL4     TIME_T OF LAST ...GET()
IPCQCTIME  DS   XL4     TIME_T OF LAST ...CTL()
IPCQTTIME  DS   XL4     TIME_T CHANGED BY TERMINATION
*-----*
*   Start of Unique data for IPCQTYPE requested
*-----*
IPCQREST   DS   0C      IPCQMSG, IPCQSHM, IPCQSEM, MAPPED MEMORY
*****
* Message Queue unique data
*****
ORG        IPCQREST
DS         0F
IPCQBYTES  DS   F       # BYTES OF MESSAGES ON QUEUE
IPCQQBYTES DS   F       MAX # BYTES OF MESSAGES ALLOWED ON QUEUE
IPCQLSPID  DS   F       PID OF LAST MSGSND()
IPCQLRPID  DS   F       PID OF LAST MSGRCV()
IPCQSTIME  DS   F       TIME_T OF LAST MSGSND()
IPCQRTIME  DS   F       TIME_T OF LAST MSGRCV()
IPCQNUM    DS   F       # OF MESSAGES ON QUEUE
IPCQRCNT   DS   F       COUNT OF WAITING MSGRCV
IPCQSCNT   DS   F       COUNT OF WAITING MSGSND
DS         0CL16      MSGRCV AND MSGSND WAITERS
DS         0CL8       MSGRCV - WAIT FOR TYPE
IPCQQRPID  DS   F       PROCESS ID
IPCQQRMSGTYPE DS   F     MESSAGE TYPE
DS         0CL8       MSGSND - WAIT FOR ROOM TO SEND
IPCQQSPID  DS   F       PROCESS ID
IPCQQSMSGLEN DS   F     MESSAGE LENGTH
DS         9CL16      MSGSND AND MSGRCV WAITERS
DS         0CL8       MESSAGES WAITING TO BE RECEIVED
IPCQQMPID  DS   F       PROCESS ID
IPCQQMMSGTYPE DS   F     MESSAGE TYPE
DS         9CL8       MESSAGES
DS         F          Reserved
DS         0D
* The 64 bit time fields will be set for either 31 or 64 bit mode
* Must define storage different, depending on how assembled
* AMODE 31
IPCQSTIME64 DS   2F     TIME64_T OF LAST MSGSND()
IPCQRTIME64 DS   2F     TIME64_T OF LAST MSGRCV()
IPCQQRMSGTYPE64 DS 20F   MSGRCV 64 BIT MSG TYPE
IPCQQMMSGTYPE64 DS 20F   MSG WAITING 64 BIT MSG TYPE
DS         CL96       Reserved for expansion
*****
* Semaphore unique data
*****
ORG        IPCQREST
DS         0F
IPCQLOPID  DS   XL4     PID OF LAST SEMOP
IPCQOTIME  DS   F       TIME_T LAST SEMOP
IPCQADJBADCNT DS   F     TERMINATION BUMPS SEM_VAL LIMITS
IPCQNSEMS  DS   FL2     NUMBER OF SEMAPHORES IN THIS SET
IPCQADJCNT  DS   FL2     NUMBER OF UNDO STRUCTURES
IPCQNCNT   DS   FL2     COUNT OF WAITERS FOR >0
IPCQZCNT   DS   FL2     COUNT OF WAITERS FOR =0
DS         0CL16      WAITERS AND ADJUSTERS
DS         0CL8       WAITER
IPCQSWPID  DS   F       PROCESS ID
IPCQSWNUM  DS   H       SEMAPHORE NUMBER
IPCQSWOP   DS   H       SEMAPHORE OPERATION
DS         0CL8       ADJUSTER
IPCQSAPID  DS   F       PROCESS ID
IPCQSANUM  DS   H       SEMAPHORE NUMBER
IPCQSAADJ  DS   H       SEMAPHORE OPERATION
DS         9CL16      WAITERS AND ADJUSTERS
DS         0D
* AMode 31

```

```

IPCQ0TIME64 DS 2F TIME64_T LAST SEMOP
DS CL360 Reserved for expansion
*****
* Shared Memory unique data *
*****
ORG IPCQREST
DS 0F
IPCQACNT DS F USE COUNT (#SHMAT - #SHMDT)
IPCQSEGSZ DS F MEMORY SEGMENT SIZE
IPCQDTIME DS F TIME_T OF LAST SHMDT()
IPCQATIME DS F TIME_T OF LAST SHMAT()
IPCQLPID DS F PID OF LAST SHMAT() OR SHMDT()
IPCQCPID DS XL4 PID OF CREATOR
*-----*
* 31 bit callers - 10 Element array of segments attached *
* Each element is the 4 byte PID followed by the 31 bit address *
*-----*
IPCQATPID DS F ATTACHED PROCESS ID
IPCQATADDRESS DS F SEGMENT ADDRESS FOR PROCESS
DS 18F MORE ATTACHED PROCESS IDS AND
* SEGMENT ADDRESS
DS 20F Reserved - match 64 bit lengths for array
IPCQDTIME64 DS 2F TIME_T OF LAST SHMDT()
IPCQATIME64 DS 2F TIME_T OF LAST SHMAT()
DS 2F Reserved - segment size in 64 bit section
DS CL344 Reserved
*****
* Mapped Memory unique data *
*****
ORG IPCQREST Mapped Memory unique data
DS 0F
IPCQMAPCPID DS F CREATOR PROCESS ID
IPCQMAPUPID DS F USER PROCESS ID
IPCQMAPTOKEN DS 2F MAP TOKEN
IPCQMAPUID DS F USER'S EFFECTIVE UID
IPCQMAPGID DS F USER'S EFFECTIVE GID
IPCQMAPFLAGS DS XL4 FLAGS
* Flags in first byte
IPCQMAPSHUT EQU X'80' SHUTDOWN OF OBJECT
IPCQBLKSZ DS F SIZE OF BLOCKS IN MEGS
IPCQBLKSINUSE DS F NUMBER OF BLOCKS IN USE
IPCQBLKSINMAP DS F NUMBER OF BLOCKS IN MAP AREA
IPCQBLKSMAPPED DS F NUMBER OF BLOCKS MAPPED
* BY THIS PROCESS
DS CL508 Reserved for expansion
*****
* Continuation of Common data *
* This next ORG gets us past the largest unique section of data *
* We need to preserve the field offsets from prior releases so *
* needed to add the rest of this common data at the end of the *
* unique data instead of within the common area defined above. *
*****
ORG
IPCQGTIME64 DS 2F TIME64_T OF LAST ...GET()
IPCQCTIME64 DS 2F TIME64_T OF LAST ...CTL()
IPCQTTIME64 DS 2F TIME64_T CHANGED BY TERMINATION
IPCQSECLABEL DS 2F SECLABEL
*****
* Overview - summary data for msgqs, semaphores, shared memory *
*****
ORG IPCQOVER Overview
DS 0F MESSAGE QUEUES
IPCQOMSGNIDS DS F Maximum number MSQs allowed
IPCQOMSGHIGHH20 DS F Most MSQs at one time
IPCQOMSGFREE DS F Number MSQs available
IPCQOMSGPRIVATE DS F Number MSQs with Ipc_PRIVATE
IPCQOMSGKEYED DS F Number MSQs with KEYS
IPCQOMSGREJECTS DS F TIMES MSGGET DENIED
IPCQOMSGQBYTES DS F MAX BYTES PER QUEUE
IPCQOMSGQMNUM DS F MAX NUMBER MESSAGES PER QUEUE
IPCQOMSGNOALC DS F # MSGSND S THAT RETURNED ENOMEM
DS F
DS 0F SEMAPHORE
IPCQOSEMNIDS DS F Maximum number SEMs allowed
IPCQOSEMHIGHH20 DS F Most SEMs at one time
IPCQOSEMFREE DS F Number SEMs available
IPCQOSEMPRIVATE DS F Number SEMs with Ipc_PRIVATE
IPCQOSEMKEYED DS F Number SEMs with KEYS
IPCQOSEMREJECTS DS F TIMES SEMGET DENIED
IPCQOSEMSNSEMS DS F MAX NUMBER OF SEMAPHORES PER SET
IPCQOSEMSNOPS DS F MAX NUMBER OPERATION IN SEMOP
IPCQOSEMSBYTES DS F STORAGE LIMIT

```

```

IPCQ0SEMCMBYTES DS F STORAGE COUNT
DS F
DS 0F SHARED MEMORY
IPCQ0SHMNIDS DS F Maximum number SHMs allowed
IPCQ0SHMHIGHH20 DS F Most SHMs at one time
IPCQ0SHMFREE DS F Number SHMs available
IPCQ0SHMPRIVATE DS F Number SHMs with Ipc_PRIVATE
IPCQ0SHMKEYED DS F Number SHMs with KEYS
IPCQ0SHMREJECTS DS F TIMES SHMGET DENIED
IPCQ0SHMSPAGES DS F MAX # PAGES PER SYSTEM LIMIT
IPCQ0SHMMPAGES DS F MAX # PAGES PER SEGMENT LIMIT - ZERO
* IF 32 BITS EXCEEDED - USE
* IPCQ0SHMMPAGES64 FOR GREATER THAN 32
* BITS
IPCQ0SHMNSEGS DS F MAX # SEGMENTS PER PROCESS LIMIT
IPCQ0SHMCPAGES DS F CURRENT # BYTES SYSTEM WIDE
* This field does not include pages for
* shared memory requests processed with
* the ipc_MEGA option
IPCQ0SHMBIGGEST DS F LARGEST SEGMENT ALLOCATED - ZERO IF
* 32 BITS EXCEEDED - USE
* IPCQ0SHMBIGGEST64 FOR GREATER THAN 32
* BITS
DS 0D
IPCQ0SHMMPAGES64 DS 2F MAX # PAGES PER SEGMENT LIMIT
IPCQ0SHMBIGGEST64 DS 2F LARGEST SEGMENT ALLOCATED
ORG ,
IPCQ#LENGTH EQU *-IPCQ Storage needed for w_getipc function
* w-getipc Command:
IPCQ#MSG EQU 1 Retrieve next message queue
IPCQ#SHM EQU 2 Retrieve next shared memory segment
IPCQ#SEM EQU 3 Retrieve next semaphore set
IPCQ#ALL EQU 4 Retrieve next member, all mechanisms
IPCQ#OVER EQU 5 Retrieve overview
IPCQ#MAP EQU 6 Retrieve mapped memory
** BPXYIPCQ End

```

## BPXYITIM — Map getitimer, setitimer structure

AMODE 64 callers use “BPXYITIM — Map getitimer, setitimer structure” on page 1202.

```

BPXYITIM ,
** BPXYITIM: getitimer and setitimer interval structure
** Used By: GTR STR
ITIM DSECT ,
** STRUCTURE OF GETITIMER (PARAMETER 2), SETITIMER (PARAMETERS 2,3)
ITIMIPAIR DS 0CL8 Initial value or value at cancel
ITIMISECONDS DS F Seconds 0-7FFFFFFF x
ITIMIMICROSEC DS 0F Microseconds 0-000F423F x
ITIMINANOSEC DS F Nanoseconds 0-369AC9FF x
ITIMRPAIR DS 0CL8 Reload Interval
ITIMRSECONDS DS F Seconds 0-2147483647 d
ITIMRMICROSEC DS 0F Microseconds 0-999999 d
ITIMRNANOSEC DS F Nanoseconds 0-999999999 d
ITIMER_REAL EQU 0 REAL TIME
ITIMER_VIRTUAL EQU 1 VIRTUAL TIME (CPU - SYSTEM)
ITIMER_PROF EQU 2 CPU TIME
ITIMER_MICRO EQU 0 1/1,000,000 of seconds
ITIMER_NANO EQU 4 1/1,000,000,000 of seconds
ITIM#LENGTH EQU 16 LENGTH THIS STRUCTURE
** BPXYITIM End

```

## BPXYMMG — Map interface for \_map\_init and \_map\_service

AMODE 64 callers use “BPXYMMG — Map Interface for \_map\_init and \_map\_service” on page 1202.

```

BPXYMMG ,
** BPXYMMG: BPX1MMI & BPX1MMS Interface Declares
** Used By: Callers of the BPX1MMI & BPX1MMS Interface
*
*****

```

```

*
*   Function Code Constants
*
*****
MMG_INIT          EQU 1
MMG_SERVICE       EQU 2
*
*****
*   Parameter list mapping for the BPX1MMI MMG_INIT call
*
*****
*
*_MMG_INIT_PARM    DSECT ,      MMG_INIT Parameter List
*_MMG_NUMBLKS      DS    F      Fullword that contains the number of
*                               blocks to be contained in the map
*                               area.
*_MMG_MEGSPERBLK   DS    F      Fullword that contains the size in
*                               megabytes of each block in the map
*                               area
*_MMG_MAPTOKEN     DS    CL8     Token for map area
*_MMG_RES01A       DS    A      Reserved for future use
*_MMG_RES01B       DS    A      Reserved for future use
*_MMG_AREAAADDR     DS    A      Fullword that contains, on input,
*                               the suggested starting address of the
*                               map area or 0. On output, this field
*                               is set to the actual map starting
*                               address.
*_MMG_INIT_PARM_LEN EQU *-*_MMG_INIT_PARM
*
*****
*   Parameter list mapping for the BPX1MMS MMG_SERVICE request
*
*   The parameter list is an array of entries, each entry having the
*   format as mapped by *_MMG_SERVICE_BLK. Each entry is a request for
*   one of the supported request types: MMG_NEWBLOCK, MMG_CONN,
*   MMG_DISCONN, MMG_CNTL or MMG_FREE. In addition, an entry can be
*   marked as inactive by setting its value to MMG_NOP, which will
*   cause the entry to be skipped. The result of a given request will
*   be reflected in the array entry.
*
*   The meaning of array entry fields is dependant on the requested
*   function. The following table defines the field meanings for each
*   of the supported functions. A field not used by a service is marked
*   N/A. Fields so marked are ignored and their value is not
*   important for the specified service. All reserved fields must be
*   zero.
*
*   Function      Field      Field usage
*   -----
*_newblock
*       *_MMG_SERVICETYPE    MMG_NEWBLOCK
*       *_MMG_SERVICEIFLAG   All bits should be zero except
*                               MMG_NOCONN may be set to one if
*                               the new block is to be allocated
*                               in the backing storage but not
*                               connected to the map area
*       *_MMG_SERVICEOFLAG   Should be zero, but not checked
*_MMG_Token         output
*_MMG_BlkJAddr      input  - 0 or address where the
*                               new block is to be
*                               allocated
*                               output - An address in the map
*                               area where the new
*                               block was allocated
*_conn
*       *_MMG_SERVICETYPE    MMG_CONN
*       *_MMG_SERVICEIFLAG   All bits should be zero
*       *_MMG_SERVICEOFLAG   Should be zero, but not checked
*_MMG_Token         input
*_MMG_BlkJAddr      input  - 0 or address where the
*                               block identified by
*                               token is to be
*                               allocated
*                               output - An address in the map
*                               area where the block
*                               was allocated
*_disconn
*       *_MMG_SERVICETYPE    MMG_DISCONN
*       *_MMG_SERVICEIFLAG   All bits should be zero except

```

```

*
* the MMG_FREE bit may be on if
* backing storage is to be
* released for the data
*
* _MMG_SERVICEOFLAG Should be zero, but not checked
* _MMG-Token N/A
* _MMG_BlklAddr input - Address of the block
* containing data to
* be disconnected
*
* _free
*
* _MMG_SERVICETYPE MMG_FREE
* _MMG_SERVICEIFLAG All bits should be zero
* _MMG_SERVICEOFLAG Should be zero, but not checked
* _MMG-Token input - Token of the data
* contained in the
* backing storage which
* is to be release
*
* _MMG_BlklAddr N/A
*
* _cntl
*
* _MMG_SERVICETYPE MMG_CNTL
* _MMG_SERVICEIFLAG All bits should be zero except
* those that define the access
* state of the data (read or
* read/write flags)
*
* _MMG_SERVICEOFLAG Should be zero, but not checked
* _MMG-Token N/A
* _MMG_BlklAddr input - Address of the block
* containing data to be
* affected by the state
* change
*
*****
*
* _MMG_SERVICE_PARM DSECT , MMG_SERVICE Parameter List
* _MMG_SERVICE_ENTRY DS 0H
* _MMG_SERVICETYPE DS FL2 Type of service requested. eg, MMG_CONN
* _MMG_SERVICEIFLAG DS BL1 Flags
* ORG _MMG_SERVICEIFLAG
* _MMG_READONLY EQU X'80' All pages of each area are to be made
* read-only
* _MMG_READWRITE EQU X'40' All pages of each area are to be made
* read-write
* _MMG_FREEBLOCK EQU X'20' The backing storage for the specified
* block is to be freed
* _MMG_NOCONN EQU X'10' The new block is to be allocated in the
* backing storage but not connected to
* the map area
* ORG _MMG_SERVICEIFLAG+L'_MMG_SERVICEIFLAG
* _MMG_SERVICEOFLAG DS BL1 Flags
* ORG _MMG_SERVICEOFLAG
* _MMG_REQFAIL EQU X'80' If on, a failure occurred on this entry
* or this entry was not processed
* ORG _MMG_SERVICEOFLAG+L'_MMG_SERVICEOFLAG
* _MMG_TOKEN DS CL8 Token for a data block
* _MMG_RES02B DS A Reserved
* _MMG_BLKADDR DS A Fullword that contains the virtual
* address of a map area block
* _MMG_MAXARRAYCOUNT EQU 1000 Maximum number of requests that can be
* in a service request array
* _MMG_SERVICE_PARM_LEN EQU *-MMG_SERVICE_PARM
*
*****
*
* BPX1MMS SERVICE Request Constants (values for field
* _MMG_SERVICETYPE)
*
*****
*
* MMG_NOP EQU 0
* MMG_NEWBLOCK EQU 1
* MMG_CONN EQU 2
* MMG_DISCONN EQU 3
* MMG_FREE EQU 4
* MMG_CNTL EQU 5
*
*****
*
** BPXYMMG End

```

## BPXYMNTTE — Map response and element structure of w\_getmntent

DSECT (MNTENTPARMDATA) is generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you might need an additional DSECT/CSECT statement to return to the current DSECT or CSECT. To get the new version of the MNTTE, set MNTTE3=YES. Users of MNTTE3=YES must initialize MNTTEHID to 'MNT3' and set MNTTEHLEN to MNTTE#LENGTH.

```

BPXYMNTTE MNTTE3=YES
** BPXYMNTTE: OpenMVS w_getmntent response structure and element
** Used By: GMN
MNTTEH          DSECT ,
MNTTEHID        DC    C'MNT3'   Eye catcher
MNTTEHSIZE      DC    A(MNTTE#LENGTH) Size of area (MNTTEH+MNTTE)
MNTTEHCUR       DC    XL8'0000000000000000'
*
*               Index of next element to return
*               - must be zero (i.e.
*               X'0000000000000000'),
*               on initial call
*               - must be left undisturbed
*               for subsequent calls
MNTTEHDEVNO     DS      F'0'   Device number - this value is
*                               specified if information about only
*                               one file system is requested
MNTTEHLEN       DS      F      Length of mntte body used
MNTTEHRES1      DS      BL8    Reserved for future - must be zero
*                               on entry
MNTTE#LENGTH    EQU    *-MNTTEH Length of header structure
*
MNTTE           DSECT ,
MNTTEBODYV1V2  DS      0F      Define V1 and V2 body size
MNTTEFSTYPE    DS      F      File system type
MNTTEFSTYPMVS  EQU      1      MVS Local File System
MNTTEFSTYPEREMOTE EQU      2      Remote File System
MNTTEFSTYPEPIPE EQU      3      Pipe file system
MNTTEFSTYPESOCKET EQU      4      Socket file system
MNTTEFSTYPEXPFS EQU      5      Cross System PFS (XPFS)
MNTTEFSTYPECSPS EQU      6      Char special streams
MNTTEFSTYPENFS EQU      MNTTEFSTYPEREMOTE
MNTTEFSMODE    DS      0F      File system mount flags
MNTTEFSMODE1   DS      B      File system mount method - byte 1
MNTTEFSMODE2   DS      B      File system mount method - byte 2
MNTTEFSMODEPRIVATE EQU    X'04' Private mount
MNTTEFSMODEUNBINDABLE EQU  X'02' Unbindable mount
MNTTEFSMODE3   DS      B      File system mount method - byte 3
MNTTEFSMODEREMOUNT EQU    X'80' Change mount mode
MNTTEFSMODEREC EQU    X'40' Recursive mount w/ bind
MNTTEFSMODEMOVE EQU    X'20' Move mount point
MNTTEFSMODEBIND EQU    X'10' Bind mount
MNTTEFSYNCHONLY EQU    X'01' File system SynchOnly specified
MNTTEFSMODE4   DS      B      File system mount method - byte 4
MNTTESECACL    EQU    X'80' Acls supported by sec product
MNTTEFSAUNMOUNT EQU    X'40' UnMount during recovery
MNTTEFSCIENT   EQU    X'20' File system is a client
MNTTEFSNOAUTOMOVE EQU    X'10' Automove allowed
MNTTEFSMODENOSSEC EQU    X'08' No Security checks enforced
MNTTEFSMODEEXPORT EQU    X'04' File system exported by DFS
MNTTEFSMODENOSUID EQU    X'02' SetUID not permitted for
*                               files in this file system
MNTTEFSMODERDONLY EQU    X'01' File system mounted read only
MNTTEFSMODERDWR EQU    X'00' File system mounted read/write
MNTTEFSDEV     DS      F      st_dev value to be returned by
*                               the stat system call for all files
*                               in this file system
MNTTEPARENTDEV DS      F      st_dev of the parent file system
MNTTEROOTINO   DS      F      ino of the mount point
MNTTESTATUS    DS      B      Status of the file system
MNTTEFILEACTIVE EQU    B'00000000' File system is active
MNTTEFILEDEAD  EQU    B'00000001' File system is dead
MNTTEFILERESET EQU    B'00000010' File system being reset
MNTTEFILEDRAIN EQU    B'00000100' File system being unmounted with
*                               drain option
MNTTEFILEFORCE EQU    B'00001000' File system being unmounted with
*                               force option
MNTTEFILEIMMED EQU    B'00010000' File system being unmounted with
*                               immmed option
MNTTEFILEENORM EQU    B'00100000' File system being unmounted with
*                               normal option

```

MNTENTIMMEDTRIED	EQU	B'01000000'	File system Umount immed fail
MNTENTQUIESCED	EQU	B'10000000'	File system is quiesced
MNTENTMNTINPROGRESS	EQU	B'10000001'	Mount in progress for
*			this file system
MNTENTASYNCHMOUNT	EQU	B'10000010'	Asynchronous mount in progres
*			for this file system
MNTENTFSDNAME	DS	CL9	DDNAME specified on mount - null
*			terminated
MNTENTFSTNAME	DS	CL9	File system type name -
*			from the FILESYSTYPE parmlib
*			statement - null terminated
MNTENTFSNAM44	DS	CL44	File system name - as a 44 byte field

Source Statement		HLASM R6.0 2016	
	ORG	MNTENTFSNAM44	
MNTENTFSNAME	DS	CL45	File system name - for PDSE/X, this
*			is the name of the PDSE/X containing
*			file system, null terminated
MNTENTPATHLEN	DS	F	length of mount point path name
MNTENTMOUNTPOINT	DS	CL1024	Name of directory where the file
*			system is mounted - (mount point
*			path name - null terminated
MNTENTJOBNAME	DS	CL8	Job name of quiesce requestor
MNTENTPID	DS	F	PID of quiesce requestor
MNTENTPARMOFFSET	DS	F	Offset of MntEntParm from MNTE
*			(Zero if none)
MNTENTPARMLEN	DS	H	Length of mount parameter
*			(Zero if none)
MNTENTSYSNAME	DS	CL8	Name of system to mount on
MNTENTQSYSNAME	DS	CL8	Name of queisce system name
MNTENTFROMSYS	DS	CL8	Filesystems to be moved from here
MNTENTRES00	DS	2B	Alignment
MNTENTRFLAGS	DS	0F	Request flags
MNTENTRFLAGS1	DS	B	Request flags - byte 1
MNTENTRFLAGS2	DS	B	Request flags - byte 2
MNTENTRFLAGS3	DS	B	Request flags - byte 3
MNTENTRFLAGS4	DS	B	Request flags - byte 4
MNTENTCHANGE	EQU	X'01'	Change f.s. server request
MNTENTNEWAUTO	EQU	X'02'	Change automove setting
MNTENTSTATUS2	DS	0F	Status of filesystem
MNTENTSTATUS2B1	DS	B	Status of filesystem - byte 1
MNTENTSTATUS2B2	DS	B	Status of filesystem - byte 2
MNTENTSTATUS2B3	DS	B	Status of filesystem - byte 3
MNTENTSTATUS2B4	DS	B	Status of filesystem - byte 4
MNTENTUNOWNED	EQU	B'00000001'	File system unowned
MNTENTINRECOVERY	EQU	B'00000010'	File system in recovery
MNTENTSUPERQUIESCED	EQU	B'00000100'	File system super quiesced
MNTENTRECYCLESTARTED	EQU	B'00001000'	PFS in recycle
MNTENTRECYCLEMOUNTING	EQU	B'00010000'	PFS recycle asynch mounting
MNTENTRECYCLENOTACTIVE	EQU	B'00100000'	PFS recycle mounts pending
MNTENTSUCCESS	DS	F	Successful moves
MNTENTREADCT	DS	F	Number of reads from filesys
MNTENTWRITECT	DS	F	Number of writes done
MNTENTDIRIBC	DS	F	Number of directory I/O blocks
MNTENTREADIBC	DS	F	Number of read I/O blocks
MNTENTWRITEIBC	DS	F	Number of write I/O blocks
MNTENTBYTESREAD	DS	BL8	Number of bytes read
MNTENTBYTESWRITTEN	DS	BL8	Number of bytes written
MNTENTFILETAG	DS	CL4	File tag (see BPXYSTAT)
MNTENTSYSLISTOFFSET	DS	F	Offset of system list
MNTENTSYSLISTLENGTH	DS	H	Length of system list
MNTENTAGGNAMELENGTH	DS	H	Length of Aggregate name
MNTENTAGGNAMEOFFSET	DS	F	Aggregate Name Offset or 0
MNTENTROSECLABEL	DS	CL8	Readonly seclabel
MNTENTBODYV3	DS	0F	Define V3 body size
MNTENTMNTSEC	DS	F	Seconds since mount
MNTENTFSTOKEN	DS	F	VfsPtr for this filesystem
MNTENTPFSNORMALSTATUSOFFSET	DS	F	Offset of Pfs normal Status
MNTENTPFSNORMALSTATUSLENGTH	DS	H	Length of Pfs normal Status
MNTENTPFSEXCPSTATUSLENGTH	DS	H	Length of Pfs exception Status
MNTENTPFSEXCPSTATUSOFFSET	DS	F	Offset of Pfs exception Status
MNTENTFSUSRMNTUID	DS	F	UID of user that mounted the file
*			system. Always 0 for privileged
*			mounts.
MNTENTFSMISC	DS	XL1	FS Misc Flags
MNTENTFSUNMNTVER	EQU	X'80'	Auto-unmount version FS
MNTENTVFSMIRROR	EQU	X'40'	vfs mirroring in progress
MNTENTSRCPATHLENGTH	DS	H	Length of Source path
MNTENTSRCPATHOFFSET	DS	F	Offset of Source path
MNTMOUNTNAMESPACE	DS	0F	Mount Namespaces
MNTMOUNTNSID	DS	CL8	Mount Namespace ID

## BPXYMODE

```

MNTEMOUNTNSREST      DS      CL24      Mount Namespace Rest
MNTEEXPANSION         DS      CL36      Expansion Room
MNTE#LENGTH          EQU      *-MNTE   Length of this structure
*
*
MNTENTPARMDATA        DSECT    ,          Mount() parameter data dsect
MNTENTPARM           DS      0C          Parameter specified with mount()
*
*
MNTENTSYSLISTINFO     DSECT    ,          Deadsys move to syslist dsect
MNTENTSYSLISTNUM      DS      H          Number of entries in the syslist
MNTENTSYSLISTFLAGS    DS      H          Flags
MNTENTSYSLISTINCL     EQU      X'0000'   Include syslist
MNTENTSYSLISTEXCL     EQU      X'0001'   Exclude syslist
MNTENTSYSLIST         DS      32CL8     System names
*
*
MNTENTAGGNAMEDSECT    DSECT    ,          At MntEntAggNameOffset if not 0
MNTENTAGGNAME         DS      0C          Aggregate Name, Null terminated
*
*
MNTENTPFNORMALSTATUS  DSECT    ,          Pfs normal status dsect
MNTENTPFNORMALSTATUS DS      0C          Pfs normal status
*
*
MNTENTPFSEXCSTATUS    DSECT    ,          Pfs exception status dsect
MNTENTPFSEXCSTATUS    DS      0C          Pfs exception status
*
* To access MNTEH, MNTE and MNTENTPARM:
* LA      RegOne,buffer      RegOne->BPX1GMN buffer and MNTEH
* USING   MNTEH,RegOne      Addressability to MNTEH
*
* LR      RegTwo,RegOne      RegTwo->MNTEH
* LA      RegTwo,MNTEH#LENGTH(RegTwo) RegTwo->MNTE
* USING   MNTE,RegTwo      Addressability to MNTENTPARMLEN
*                                     and MNTENTPARMOFFSET
*
* ICM      RegThree,15,MNTENTPARMOFFSET Load offset from start of
*                                     entry (i.e. start of MNTE)
* BZ      SkipParm          If zero, skip processing parm
* ALR      RegThree,RegTwo   RegTwo->MNTE,
*                                     RegThree=MNTENTPARMOFFSET
*                                     RegThree->MNTENTPARMDATA (after)
* USING   MNTENTPARMDATA,RegThree Addressability to MNTENTPARMDATA
*
** BPXYMNTE End

```

## BPXYMODE – Map the mode constants

```

                BPXYMODE
** BPXYMODE: Mode constants specified on system calls
** Used By: CHM FCM MKD MKN OPN UMK
S_MODE          DSECT    ,
                DS      0F
*
S_TYPE          DS      B          File types, mapped by BPXYFTYP
*
S_MODE3B        DS      0XL3      All flag bytes
S_RES01         DS      0BL.8     Reserved
S_MODE1         DS      B          Flag byte 1 - reserved
*
S_RES02         DS      0BL.4     Reserved
S_MODE2         DS      B          Flag byte 2
*
S_ISUID         EQU      X'08'     Set user ID on execution
S_ISGID         EQU      X'04'     Set group ID on execution
S_ISVTX         EQU      X'02'     Sticky Bit: For executables, look
*                                     first in normal MVS search order
*                                     For directories, deletion rstd
*                                     to owner or superuser.
*
*
S_IRWXU1        EQU      X'01'     All permissions for user - part I
S_IRUSR         EQU      X'01'     Read permission
*
S_MODE3         DS      B          Flag byte 3
*
S_IRWXU2        EQU      X'C0'     All permissions for user - Part II
S_IWUSR         EQU      X'80'     Write permission

```



```

S_IXUSR      EQU   X'40'  Search (if a directory) or
*              execute (otherwise) permission
*              Group flags
S_IRWXG      EQU   X'38'  All permissions for group
S_IRGRP      EQU   X'20'  Read permission
S_IWGRP      EQU   X'10'  Write permission
S_IXGRP      EQU   X'08'  Search (if a directory) or
*              execute (otherwise) permission
*              Other flags
S_IRWXO      EQU   X'07'  All permissions for other
S_IROTH      EQU   X'04'  Read permission
S_IWOTH      EQU   X'02'  Write permission
S_IXOTH      EQU   X'01'  Search (if a directory) or
*              execute (otherwise) permission
S_MODE#LENGTH EQU   *-S_MODE Length this structure
** BPXYMODE End

```

## BPXYMSG — Map interprocess communication message queues

DSECT (MSGBUF) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. Default for the message size is 100 bytes. Specify VARLEN= to override this value.

AMODE 64 callers use “BPXYMSG — Map interprocess communication message queues” on page 1204.

```

                BPXYMSG
** BPXYMSG: Interprocess Communication Message Queue Structure
** Used By: msgctl
MSGID_DS       DSECT ,      message queue structure
MSG_PERM       DS          CL(IPC#LENGTH) Mapped by BPXYIPCP
MSG_QNUM       DS          F      # of messages on queue
MSG_QBYTES     DS          F      max bytes allowed on queue
MSG_LSPID      DS          F      process ID of last msgsnd()
MSG_LRPID      DS          F      process ID of last msgrcv()
MSG_STIME      DS          F      time of last msgsnd()
MSG_RTIME      DS          F      time of last msgrcv()
MSG_CTIME      DS          F      time of last change get/ctl
MSG#LENGTH     EQU *-MSGID_DS Length of this DSECT
MSGBUF         DSECT ,      Message buffer - msgsnd, msgrcv
MSG_TYPE       DS          F      Message type
MSG_MTEXT      DS          CL100 Message text
MSGB#LENGTH    EQU *-MSGBUF Length of this DSECT
MSGXBUF        DSECT ,      Message buffer - msgxrcv
MSGX_MTIME     DS          F      time message sent
MSGX_UID       DS          F      sender's effective UID
MSGX_GID       DS          F      sender's effective GID
MSGX_PID       DS          F      sender's PID
MSGX_TYPE      DS          F      Message type
MSGX_MTEXT     DS          CL100 Message text
MSGX#LENGTH    EQU *-MSGXBUF Length of this DSECT
* Flag bits - msgrcv (also IPC_NOWAIT
MSG_NOERROR    EQU        4      No error if big message.
MSG_INFO       EQU        8      Use MSGXBUF not MSGBUF format
** BPXYMSG End

```

## BPXYMSGF — Map the message flags

BPXYMSGF is used by send(), recv(), sendto(), recvfrom(), sendmsg() and recvmsg()

```

** BPXYMSGF: Socket MSG_* flags
** Used By: SND RCV STO RFM SMS RMS SRX AIO
AIF ('&DSECT' EQ 'NO').B411
MSG_FLAGS      DSECT ,
AGO .C411
.B411 ANOP ,
MSG_FLAGS      DS          0F
.C411 ANOP ,
MSG_FLAGS1     DS          B      I_flags - byte 1
MSGFHIGH       EQU   X'80'  DO NOT USE THIS BIT! *
*              MSG_FLAGS must never be < 0
*
MSG_ACK_GEN     EQU   X'40'  Generate a UDP 'ACK packet' *

```

```

*                                     automatically to the originator if
*                                     an incoming UDP packet arrives
*                                     This flag is no longer supported by z/OS TCPIP.
*
MSG_ACK_TIMEOUT      EQU   X'20'  The caller expects an incoming UDP
*
*                                     packet within the "standard ACK
*                                     time interval". Return to caller
*                                     with an EINTR return code if no
*                                     incoming UDP packet arrives
*                                     within this time interval.
*                                     This flag is no longer supported by z/OS TCPIP.
*
MSG_ACK_EXPECTED     EQU   X'10'  (Used along with MSG_ACK_TIMEOUT)
*                                     The incoming packet is expected to
*                                     be an ACK. If the ACK arrives,
*                                     the caller does not need to be
*                                     activated to process it.
*                                     Instead, the protocol will just
*                                     cancel the timeout and let the
*                                     application wait for the real data
*                                     to arrive.
*                                     This flag is no longer supported by z/OS TCPIP.
SPACE ,
MSG_FLAGS2           DS      B
MSG_flags - byte 2
*
SPACE ,
MSG_FLAGS3           DS      B      MSG_flags - byte 3
MSG_EOF              EQU   X'80'  Close socket after the send.
*                                     On the send function only - Requests that the socket
*                                     be closed after all data has been transmitted.
*
SPACE ,
MSG_FLAGS4           DS      B      MSG_flags - byte 4
*
MSG_CONNTERM         EQU   X'80'  Complete when connection ends.
*                                     Requests that a receive-type function completes only
*                                     when a TCP socket connection is terminated.
*                                     - The buffer length specified on the operation must be 0,
*                                     thus there is no data associated with this receive and
*                                     the other MSG_FLAGS may not be used. This operation is
*                                     only for connection termination notification.
*                                     - Any other normal outstanding receive-type requests will
*                                     also be completed at connection termination and these
*                                     completions may be running in parallel with that of
*                                     the MSG_CONNTERM request.
*
MSG_WAITALL          EQU   X'40'  Wait until all data returned
*                                     Requests that a receive-type function block until the
*                                     full amount of data requested can be returned.
*                                     The function may return a smaller amount of data if a
*                                     signal is caught, the connection is terminated, an error
*                                     is pending or SO_RCVTIMEO is set and the timer expires.
*
MSG_CTRUNC           EQU   X'20'  Control data was truncated.
*                                     An output flag for recvmsg, returned in the MsgHFlags
*                                     field of the MSGH structure.
MSG_TRUNC            EQU   X'10'  Normal data was truncated.
*                                     An output flag for recvmsg, returned in the MsgHFlags
*                                     field of the MSGH structure.
*
MSG_EOR              EQU   X'08'  Terminate a record. @D2A
*                                     Only supported for FRCA enabled TCP sockets.
*
MSG_DONTRROUTE       EQU   X'04'  Send without network routing.
*                                     Bypass normal routing for one send-type function request.
*                                     Usually used only by diagnostic or routing programs.
*
MSG_PEEK             EQU   X'02'  Peek at incoming data.
*                                     The data is returned but not consumed, so that a
*                                     subsequent receive-type function will see the same data.
*
MSG_OOB              EQU   X'01'  Send/Receive out of band data.
*                                     Sends out-of-band data for send, sendto or sendmsg.
*                                     Requests out-of-band data for recv, recvfrom or recvmsg.
*                                     Also, is an output flag for recvmsg indicating that
*                                     out-of-band data was received. This is returned
*                                     in the MsgHFlags field of the MSGH structure,
MSG#LENGTH           EQU   *-MSG_FLAGS Length of this structure

```

```
** BPXYMSGF End
```

## BPXYMSGH — Map the message header

BPXYMSGH is used by the sendmsg and recvmsg syscalls. AMODE 64 callers use “BPXYMSGH — Map the message header” on page 1205.

```

                BPXYMSGH      ,
** BPXYMSGH: MSGH system call structure
** Used By: SendMsg / RecvMsg
MSGH            DSECT      ,
MSGHBEGIN       DS         0D
* ----- 32-Bit Version
MSGHNAMEPTR     DS         A(0)  Pointer to a structure that contains
*                               the recipient's address.
MSGHNAMELEN     DS         F'0'  Buffer length.
MSGHIOVPTR      DS         A(0)  Pointer to an array of IOVEC buffers.
MSGHIOVNUM      DS         F'0'  Number of elements in IOVEC array.
MSGHCONTRLPTR   DS         0AL4  Pointer to ancillary data buffer
MSGHACCRIGHTSPTR DS         A(0)  Pointer to access rights buffer.
MSGHCONTROLLEN  DS         0FL4  Length of ancillary data buffer
MSGHACCRIGHTSLEN DS         F'0'  Access rights buffer length.
MSGHFLAGS       DS         F'0'  Output flags on received message
*
*   Constants
*
MSGH#LENGTH     EQU      *-MSGH  Length of MsgH
*
CMMSGPTR        DS         A(0)  CMsg pointer
*
CMMSGHDR        DSECT      ,
CMSGLEN         DS         F'0'  Length, including header
CMSGLEVEL       DS         F'0'  Level
CMSGTYPE        DS         F'0'  Type
CMSGDATA        DS         0C    Data
*
*   Constants
*
SCM_RIGHTS      EQU      1       Access Rights
SCM_SECINFO     EQU      16386   Security Information
*
** BPXYMSGH End

```

## BPXYMSGX — Map the message header

BPXYMSGX is used by the srx\_np() syscall. BPXYMSGX uses constants defined by mapping macro IVTBUFL.

```

                IVTBUFL      ,
IVTBUFL DSECT      ,
BUFL_VERSION    DS         X      BUFFER DESCRIPTOR
BUFL_VERSIONC   EQU      X'00'    VERSION 0
BUFL_SOURCE     DS         X      BUFFER SOURCE
BUFL_CECSA      EQU      X'80'    INDICATES THAT THE STORAGE
*                               IS IN CSM ECSA
BUFL_CDSPACE    EQU      X'40'    INDICATES THAT THE STORAGE
*                               IS IN CSM DATA SPACE
BUFL_UDSPACE    EQU      X'20'    INDICATES THAT THE STORAGE
*                               IS IN A USER DATA SPACE
BUFL_USTOR      EQU      X'10'    INDICATES THAT THE STORAGE
*                               IS A USER'S STORAGE OTHER THAN
*                               A DATA SPACE
BUFL_HVCOMM     EQU      X'08'    Indicates that the storage
*                               is in a HVCOMM
*
BUFL_TYPE       DS         X      BUFFER TYPE
BUFL_FIXED      EQU      X'80'    INDICATES THAT THE STORAGE IS
*                               IN A GUARANTEED TO BE FIXED
*                               STATE
BUFL_PAGEABLE   EQU      X'40'    INDICATES THAT THE STORAGE IS
*                               IN A GUARANTEED TO BE PAGEABLE

```

```

*
BUFL_PAGEELIG EQU X'20' STATE
* INDICATES THAT THE STORAGE IS
* ELIGIBLE TO BE PAGEFREED BY
* CSM
DS XL1 RESERVED
BUFL_TOKEN DS XL12 CSM BUFFER TOKEN
BUFL_ALET DS F DATA SPACE ALET
BUFL_ADDR64 DS AD 64-bit buffer address
BUFL_ADDR DS A POINTER TO BUFFER
BUFL_SIZE DS F THE SIZE OF THE ALLOCATED BUFFER
* ON GET_BUFFER REQUESTS, THE DATA
* LENGTH ON COPY_DATA REQUESTS
BUFL_END DS 0F END OF IVTBUFL
        BPXYMSGX ,
** BPXYMSGX: MSGX system call structure
** Used By: BPX1SRX
MSGX DSECT ,
*
MSGXNAMEPTR DS A /* PTR TO SOCKADDR BUFFER */
MSGXNAMELEN DS F /* LENGTH OF SOCKADDR BUFFER */
MSGXFLAGS DS 0BL4 /* SRX CONTROL FLAGS */
MSGXFLAGS1 DS BL1
MSGXFLAGS2 DS BL1
MSGXFLAGS3 DS BL1
MSGXFLAGS4 DS BL1 /* FLAGS ARE IN THE 4TH BYTE
MSGX_CECSA EQU X'02' /* RECV IN ECSA BUFFERS */
MSGX_CDSpace EQU X'01' /* RECV IN DATA SPACE BUFFERS */
MSGXMSGFLAGS DS BL4 /* MSG * FLAGS, SEE BPXYMSGF */
MSGXDATALEN DS F /* MAX/MIN DATA TO RECEIVE */
MSGXTCB DS A /* TCB TO OWN RECEIVE BUFFERS */
MSGXERRIOVX DS F /* SEND IOVX ELEMENT IN ERROR */
MSGXERRDATA DS F /* AMOUNT SENT FROM LAST BUFFER */
MSGXIVTBUFLOFFSET DS F /*1ST BUFF APPL STILL OWNS
DS CL4 RESERVED
MSGXCONTROLPTR DS A Ancillary Data buffer
MSGXCONTROLLEN DS F Length of ancillary data
MSGXIOVX DS CL(BUFL_END-IVTBUFL) IVTBUFL FOR IOVX ARRAY
MSGXEND EQU *
*
MSGX#LEN EQU MSGXEND-MSGX
*
* IOVX - ARRAY OF IVTBUFL BUFFER DESCRIPTIONS
*
IOVX DSECT , /* DESCRIBED BY MSGXIOVX */
IOVXBUFL DS 0CL(BUFL_END-IVTBUFL) ARRAY ELEMENT
*
* Constants
*
MSGX_SEND EQU 0 /* BPX1SRX DIRECTION PARAMETER: */
MSGX_RECV EQU 1 /* SEND OPERATION */
/* RECEIVE OPERATION */
** BPXYMSGX End

```

## BPXYMTM — Map the modes for mount and unmount

```

        BPXYMTM ,
** BPXYMTM: File system mount/unmount modes
** Used By: MNT UMT
MTM DSECT ,
MTM1 DS B Flag byte 1
MTMRO EQU X'80' Mount file set read-only
MTMRDWR EQU X'40' Mount file set read/write
MTMDDNAME EQU X'20' FileSet is a DDName
MTMUMOUNT EQU X'10' This is a normal unmount request.
* If no one is using any of the files
* in the named filesystem, the unmount
* will be done. Otherwise, the request
* will be rejected.
MTMIMMED EQU X'08' This is an unmount immediate request.
* The filesystem will be unmounted
* immediately, forcing any users of any
* files in the named filesystem to fail.
* All data changes that were made up to
* the time of the request will be saved.
* If there is a problem saving the data,
* the unmount request will fail.
MTMFORCE EQU X'04' This is an unmount force request.

```

```

*
* The filesystem will be unmounted
* immediately, forcing any users of any
* files in the named filesystem to fail.
* All data changes that were made up to
* the time of the request will be saved.
* If there is a problem saving the data,
* the request will continue and data may
* be lost. Since data may be lost,
* before a forced request will be
* allowed, a previous immediate unmount
* request must have been attempted, or
* the request will be rejected.
MTMDRAIN      EQU    X'02' This is an unmount drain request.
* The requestor is willing to wait for
* all uses of this filesystem to be
* normally terminated and the
* filesystem to be unmounted.
MTMRESET      EQU    X'01' This is a reset unmount request. This
* will allow a previous unmount drain
* request to be stopped.
MTM2          DS      B      Flag byte 2
MTM2RES80     EQU    X'80' Must not be used
MTM2RES40     EQU    X'40' Must not be used
MTMTERMUNMOUNT EQU    X'20' Unmount from PFS term
MTMSAMEMODE    EQU    X'10' Remount in same mode
MTMMNTINCOMP  EQU    X'08' Mount is incomplete
MTMUNQSEFORCE  EQU    X'04' Force this unquiesce request, even
* if the requester process is not
* the process that made the quiesce
* request.
MTM2RES02     EQU    X'02' Must not be used
MTM2RES01     EQU    X'01' Must not be used
MTM3          DS      B      Flag byte 3 - reserved
MTM3RES80     EQU    X'80' Must not be used
MTM3RES40     EQU    X'40' Must not be used
MTM3RES20     EQU    X'20' Must not be used
MTM3RES10     EQU    X'10' Must not be used
MTM3RES08     EQU    X'08' Must not be used
MTMNOSUID     EQU    X'04' Dont allow setuid
MTMSYNCHONLY  EQU    X'02' Mount must be completed
* synchronously. That is, mount()
* must not return +1
MTMREMOUNT    EQU    X'01' Change attributes of mounted file
* system
MTM4          DS      B      Flag byte 4 - reserved
MTMNOSEC      EQU    X'80' NoSecurity option
MTM4RES40     EQU    X'40' Must not be used
MTM4RES20     EQU    X'20' Must not be used
MTMAMOVE      EQU    X'10' Automove option
* BPX2MNT/BPX4MNT use only
MTMAUNMOUNT   EQU    X'08' UnMount during recovery
* BPX2MNT/BPX4MNT use only
MTM4RES08     EQU    X'08' Must not be used
MTM4RES04     EQU    X'04' Must not be used
MTMDETACH     EQU    X'04' Detach the file system
MTM4RES02     EQU    X'02' Must not be used
MTM#LENGTH    EQU    *-MTM Length of this structure
*
* MTM#MOUNTTEST defines the valid MTM bit settings for the BPX1MNT
* interface
MTM#MOUNTTEST EQU    X'C0000680'
** BPXYMTM End

```

## BPXYOCRT — Map the OE certificate support structure

AMODE 64 callers use “BPXYOCRT — Map the OE certificate support structure” on page 1206.

```

          BPXYOCRT
** BPXYOCRT: OE Certificate support structure
** Used By: TLS
OCRT      DSECT ,
OCRTTYPE  DS      F          type of certificate attached
OCRTUSERID DS      CL9       MVS userid, null terminated, input/output
          DS      CL3       reserved
OCRTCLEN  DS      F          length of certificate associated with type
OCRTCPtr  DS      A          31-Bit ptr to the actual certificate
OCRT_LEN  EQU    *-OCRT

```

```
OCRT_X509 EQU 1      Certificate type X509
** BPXYOCRT End
```

## BPXYOEXT – Map the common external control block

DSECT=NO is not allowed. The storage belongs to z/OS UNIX.

```

** BPXYOEXT: Common External Control Block
**
** The address of BPXYOEXT control block can be obtained as follows:
**
**      L      14,16(0,0)      GET CVT ADDRESS
**      L      15,140(0,14)    GET ECVT ADDRESS
**      L      14,244(0,15)    GET BPXYOEXT ADDRESS
**
** OEXT user exit support:
**
** When the kernel detects that the OEXTUSEREXIT address is
** non-zero, control will be given to this exit on:
**
**      1) Successful completion of the GETPWNNAME service and
**         specified name matches invoking userid.
**
**           Parm 1 = 4 byte function code set to OEXT#UEGETPWNNAME
**           Parm 2 = 4 byte length of Current Working Directory
**           Parm 3 = N byte Current Working Directory
**
** Purpose of call is to allow exit to examine/change CWD.
** Length of CWD must remain the same.
**
** User Exit will be given control in supervisor state key zero.
**
** Input:
**      Register 1 = Parmlist address ---> | Parm 1 addr |
**                                           | Parm 2 addr |
**                                           | Parm N addr |
**                                           | parm
**
**      Bit 31 on in last parm
**
**      Register 13 = Save area address
**      Register 14 = Return address
**
** Output:
**      Possible modification of CWD, length must not change.
**
** Exit will be passed a 64 bytes save area in Register 13. It is
** the responsibility of the User Exit to save all registers upon
** entry and to restore all registers before return.
**
** *OEXT          DSECT ,
** OEXT           DS      0D
** OEXTID         DS      CL4      Eye catcher
** OEXTSP         DS      FL1      Subpool of this structure
** OEXTLEN        DS      FL3      Length this structure
** OEXTFLGS       DS      0CL4
** OEXTFLG1       DS      B
** OEXTF1DEFSEG   EQU      X'80'   OMVS default segment exists
**              EQU      X'40'   Reserved for future use
** OEXTSYSPLEXACTV EQU      X'20'   OMVS is a member of sysplex
** OEXTNOACCT     EQU      X'10'   Don't allow user to change acct-info
**
** OEXTSETIP@     EQU      X'08'   Deamons should set IP@ of client
**
** OEXTGRPIDVALID EQU      X'04'   If on, OextTtyGrpId is valid
** OEXTSERVICESAVAILABLE EQU    X'02' IF ON, z/OS UNIX Services Available
** OEXTFLG2       DS      B
** OEXTCINETCONF  EQU      X'80'   Cinet is configured
** OEXTIPV6       EQU      X'40'   IPV6 Configured & Supported
** OEXTIPV6CONF   EQU      X'20'   IPV6 is Configured
** OEXTFLG3       DS      B
** OEXTUMASKCONF  EQU      X'80'   UMASK specified in BPXPRM
** OEXTFLG4       DS      B
** OEXTSTARTM     DS      BL.64    Time z/OS UNIX was started

```

```

*
OEXTSMFP      DS      A      (TOD format)
*              Pointer to the SMF Process accounting
*              data collection routine. This offset
*              must not change. It is hardcoded in
*              executable macro BPXESMF.
OEXTLVL       DS      0XL4   z/OS UNIX Release level
*              indicator
*              (Multiple bits may be set, e.g.
*              OEXT1120 and OEXT1130 will both
*              be set)
OEXTLV0       DS      XL1    Byte 0 of OEXTLVL
*              Reserved
OEXT1120      EQU     X'80'   HOM1120 functions are present
OEXT1130      EQU     X'40'   HOM1130 functions are present
OEXT1130IPC   EQU     X'20'   HOM1130 functions InterProcess
*              Communication (IPC), NFSC, directory
*              sticky bit, lchown UID=-1, are
*              present.
OEXT1140      EQU     X'08'   HOM1140 functions are present
OEXT1150      EQU     X'04'   HBB6603 functions are present. Future
*              z/OS UNIX release indicators will
*              not be added since z/OS UNIX is
*              now an integral part of the BCP.
OEXTTECFs     EQU     X'02'   The Thli ECB communication function is
*              supported
OEXTTAFs      EQU     X'01'   The MvsThreadAffinity function is
*              supported
OEXTLV1       DS      XL1    Byte 1 of OEXTLVL
OEXTPQG       EQU     X'80'   The pthread_quiesce_and_get_np
*              function (BPX1PQG) is supported
OEXTMMAPEXTKEY EQU     X'40'   The mmap extended key function
*              is supported
OEXTAPPCBPXEXIT DS      XL2   Bytes 2-3 of OEXTLVL
*              DS      A      Pointer to the z/OS UNIX Exit
*              for APPC Processing.
OEXTTIXP      DS      A      Pointer to z/OS UNIX timer exit.
*              Invoked by IEAVLEXT.
OEXTRUNOPTSPTR DS      A      Pointer to the RUNOPTS() string
*              specified at IPL time.
OEXTRUNOPTSLEN DS      FL4    Length of the RUNOPTS() string
*              specified at IPL time.
OEXTBPXWLMEXIT DS      A      Pointer to z/OS UNIX Exit for WLM
*              IWMUWON timeout processing
OEXTUSEREXIT  DS      A      Pointer to z/OS UNIX User Exit
OEXTMSGACBRtn DS      A      Pointer to JESYSMSG ACB PUT rtn
OEXTPARMLIBPTR DS      A      Pointer to Parmlib Data Area
OEXTARG_MAX   DS      FL4    Sysconf ARG_MAX value
OEXTCLK_TCK   DS      FL4    Sysconf CLK_TCK value
OEXTTZNAME_MAX DS      FL4    Sysconf TZNAME_MAX value
OEXTJOB_CONTROL DS      FL4    Sysconf JOB_CONTROL value
OEXTVERSION   DS      FL4    Sysconf VERSION value
OEXT2_CHAR_TERM DS      FL4    Sysconf CHAR_TERM value
OEXTTTYGRP_ID DS      FL4    sysconf TTY_GROUP value
*              DS      1CL0008 Reserved
*              DS      0D      Ensure end on double word boundary
OEXT#LENGTH   EQU     *-OEXT Length of this structure
OEXT#UEGETPWNAME EQU     1    Function code indicating user exit
*              called from getpwnam
** BPXYOEXT End

```

## BPXYOPNF — Map flag values for open

```

O_FLAGS1      DS      B      Open flags - byte 1
OPNFHIGH      EQU     X'80'   DO NOT USE THIS BIT!
*              O_FLAGS must never be < 0
SPACE ,
O_FLAGS2      DS      B      Open flags - byte 2
OPNFEXEC      EQU     X'80'   Execute access requested -
O_PATH        EQU     X'08'   Opening a path @D9A
O_PRIVPROGCK  EQU     X'04'   Privileged program checking @01A
*              authorization required for use
SPACE ,
O_FLAGS3      DS      B      Open flags - byte 3
O_DIRECTORY   EQU     X'80'   Opening directory @D9A
O_NOFOLLOW    EQU     X'40'   No symlink final component @D9A
O_DIRECT      EQU     X'20'   Pipe Packet Support @D8A
O_CLOEXEC     EQU     X'10'   Close on exec @P6A

```

## BPXYPCF

```
O_NOLARGEFILE EQU X'08' Large Files not allowed @D5A
O_LARGEFILE EQU X'04' Ignored @D5A
O_ASYNC SIG EQU X'02' An asynchronous signal may occur @D4A
O_SYNC EQU X'01' Force synchronous updates @D3A
SPACE ,
O_FLAGS4 DS B Open flags - byte 4
O_CREXCL EQU X'C0' Create file (fails if file exists
O_CREAT EQU X'80' Create file
O_EXCL EQU X'40' Exclusive flag (file must not exist)
O_NOCTTY EQU X'20' Not a controlling terminal
O_TRUNC EQU X'10' Truncate flag
O_APPEND EQU X'08' Set offset to EOF on write
O_NONBLOCK EQU X'04' Don't block this file
FNDELAY EQU X'04' Don't block this file @D2A
O_RDWR EQU X'03' Open for Read and Write
O_RDONLY EQU X'02' Open for Read Only
O_WRONLY EQU X'01' Open for Write Only
O_ACCMODE EQU X'03' Mask for file access modes
O_GETFL EQU X'0F' Mask for file access modes and
* file status flags together @P4A
SPACE ,
OPNF#LENGTH EQU *-O_FLAGS Length of this structure
O_GETFL2 EQU 532480 Mask for O_PATH and O_DIRECT @P6A
```

## BPXYPCF — Command values for pathconf and pathconf

BPXYPCF is composed only of EQUates. DSECT= is allowed but ignored.

```
BPXYPCF
** BPXYPCF: Command values
** Used By: FPC PCF
PC_CHOWN_RESTRICTED EQU 1 _POSIX_CHOWN_RESTRICTED option
PC_LINK_MAX EQU 2 LINK_MAX option
PC_MAX_CANON EQU 3 _POSIX_MAX_CANON option
PC_MAX_INPUT EQU 4 _POSIX_MAX_INPUT option
PC_NAME_MAX EQU 5 NAME_MAX option
PC_NO_TRUNC EQU 6 _POSIX_NO_TRUNC option
PC_PATH_MAX EQU 7 PATH_MAX option
PC_PIPE_BUF EQU 8 PIPE_BUF option
PC_VDISABLE EQU 9 _POSIX_VDISABLE option
PC_ACL EQU 10 _PC_ACL option
PC_ACL_ENTRIES_MAX EQU 11 _PC_ACL_ENTRIES_MAX
PC_CASE EQU 100 Case Flags
*
PCFGMAX EQU 11 Max _POSIX_ value
*
* Pathconf Case Flags - vn_pathconf(PC_CASE) returned value
PCCASEFLAGS DSECT ,
DS XL3
PCCASEFLGSBYTE DS XL1
PCCASEINSENSITIVE EQU X'02' 0=SENSITIVE,1=NOT
PCCASENONPRESERVING EQU X'01' 0=PERSERVING,1=NOT
*
* Pathconf File Group - for v_pathconf(BPX1VPC)
PCFG DSECT ,
DS F LINK_MAX
PCFGNAME_MAX DS F NAME_MAX
PCFGPCFLAGS DS XL1 FLAGS:
PCFGNOTRUNC EQU X'80' POSIX_NO_TRUNC
PCFGCHOWNRSTD EQU X'40' CHOWN RESTRICTED
PCFGCASEINSENSITIVE EQU X'20' 0=SENSITIVE,1=NOT
PCFGCASENONPRESERVING EQU X'10' 0=PERSERVING,1=NOT
PCFGSEACAL EQU X'08' 0=ACLSUPPORT,1=NONE
DS XL3
*
** BPXYPCF End
```

## BPXPEDB — Mapping of process exit data block

```
BPXPEDB
PEDB DSECT , PEDB - Process Exit Data Block
PEDBEYE DS 1CL0004 Eye catcher - 'PEDB'
PEDBLENGTH DS 1FL2 Length of structure
PEDBVERSION DS 1FL1 Version number
```



```

PEDBEXITPOINTID DS 1FL1 Unique value identifying exit point, these X
constants are defined below
PEDBFLAGS DS 0FL4 Flags
PEDBCREATEDVIAFLAGS DS 0CL0001 Bits indicating what the process is X
being created via
PEDBVIAFORK EQU X'80' On = process is being created via fork()
PEDBVIAspawn EQU X'40' On = process is being created via spawn()
PEDBVIAATTEEXEC EQU X'20' On = process is being created via X
attach_exec()
PEDBVIAATTEEXECMVS EQU X'10' On = process is being created via X
attach_execmvs()
PEDBVIA1STCALLABLE EQU X'08' On = process is being created via the 1st X
callable service from a non-z/OS UNIX address space
ORG PEDBCREATEDVIAFLAGS+X'00000001'
PEDBFLAGS2 DS 0CL0001 2nd flag byte
PEDBVIAMEMTERM EQU X'80' On = process is being terminated via memterm
PEDBVIAABTERM EQU X'40' On = process is being terminated via abterm
ORG PEDBFLAGS2+X'00000001'
PEDBFLAGS3 DS 1CL0001 3rd flag byte
PEDBFLAGS4 DS 1CL0001 4th flag byte
PEDBUNIQUEID DS 1BL8 A Unique Id identifying this process's set of X
exits. This Id is the same starting at the X
pre-process initialization exit all the way X
to the pre-process term exit. It also happens X
to be TOD when the pre-process initial- X
ization exit was called.
*
* *****
* *
* * Information specific to Initiator of the new process *
* * (creator) This section is filled out ONLY when the following*
* * exits hit: BPX_PREPROC_INIT - pre-process initialization *
* * BPX_POSPROC_INIT - post process initialization *
* * BPX_IMAGE_INIT - process image change This section is NOT *
* * filled out by the following exits: BPX_PREPROC_TERM - *
* * pre-process termination *
* *
* *****
*
*
PEDBCREATORINFO DS 0CL0164
PEDBCREATORPROCID DS 1FL4 Process ID initiating New process
PEDBCREATORASID DS 1FL2 ASID of initiating new process
PEDBCREATORUSERIDLEN DS 1FL1 Length of the Userid initiating the new X
process
PEDBCREATORALIASLEN DS 1FL1 Length of the Alias initiating the new X
process
PEDBCREATORPROGNAMELEN DS 1FL2 Length of the Program Name initiating X
new process
DS 1FL2 Reserved
PEDBCREATORJOBNAME DS 1CL0008 Jobname initiating the new process
PEDBCREATORUSERID DS 1CL0008 Userid initiating the new process
PEDBCREATORALIAS DS 1CL0008 Alias initiating the new process
PEDBCREATORPROGNAME DS 1CL0128 Program Name of the initiating new X
process
*
* *****
* *
* * New Process / Terminating Process Information (child) This *
* * section is filled out ONLY when the following exits hit: *
* * BPX_POSPROC_INIT - post process initialization *
* * BPX_IMAGE_INIT - process image change BPX_PREPROC_TERM - *
* * pre-process termination This section is NOT filled out by *
* * the following exits: BPX_PREPROC_INIT - pre-process *
* * initialization *
* *
* *****
*
*
PEDBNEWINFO DS 0CL0164
PEDBTERMINFO DS 0CL0164
PEDBNEWPROCID DS 0FL4 Process ID of New process
PEDBTERMPROCID DS 1FL4 Process ID for the terminating process
PEDBNEWASID DS 0FL2 ASID of new process
PEDBTERMASID DS 1FL2 ASID of the terminating process
PEDBNEWUSERIDLEN DS 0FL1 Length of the Userid of the new process
PEDBTERMUSERIDLEN DS 1FL1 Length of the Userid of the terminating X
process
PEDBNEWALIASLEN DS 0FL1 Length of the Alias of the new process
PEDBTERMALIASLEN DS 1FL1 Length of the Alias of the terminating X
process
PEDBNEWPROGNAMELEN DS 0FL2 Length of the Program Name of the new X

```

```

                                process
PEDBTERMPROGNAMELEN DS 1FL2 Length of Program Name of the terminating X
                                process
                                Reserved
PEDBNEWJOBNAME DS 0CL0008 Jobname of new process
PEDBTERMJOBNAME DS 1CL0008 Jobname of terminating process
PEDBNEWUSERID DS 0CL0008 Userid of the new process
PEDBTERMUSERID DS 1CL0008 Userid of the terminating process
PEDBNEWALIAS DS 0CL0008 Alias of the new process
PEDBTERMALIAS DS 1CL0008 Alias of the terminating process
PEDBNEWPROGNAME DS 0CL0128 Program Name of the new process
PEDBTERMPROGNAME DS 1CL0128 Program Name of the terminating process
*
* *****
* *
* *
* *****
*
*
                                DS 1CL0064 Reserved for future use
PEDBVER1LEN DS 0C End of Version 1
PEDB#ID EQU C'PEDB' Eye catcher
PEDB#VER EQU 1 Current version of this control block
PEDB#VER01 EQU 1 Version 1 of control block
PEDB#LEN01 EQU 412 Version 1 of PEDB control block len
PEDB#LEN EQU 412 Length of PEDB
*
* Constants to fill in PEDBExitPointId field
*
*
PEDB_BPX_PREPROC_INIT EQU 1 Identifies that this this structure was X
                                built for the pre-process initiation exit
PEDB_BPX_POSPROC_INIT EQU 2 Identifies that this this structure was X
                                built for the post process initiation exit
PEDB_BPX_IMAGE_INIT EQU 3 Identifies that this this structure was X
                                built for the process image change exit
PEDB_BPX_PREPROC_TERM EQU 4 Identifies that this this structure was X
                                built for the pre-process termination
PEDB_LEN EQU *-PEDB

```

## BPXYPGPS — Map the response structure for w\_getpsent

VARLEN accepts three operands. Operands omitted (like the first) default to the maximum needed. Use zero if the associated field is not needed.

VARLEN describes the number of bytes to map the following:

1. Controlling TTY name and its length
2. Path name and its length
3. Command and its length

```

                                BPXYPGPS
VARLEN=(1028,1028,1028)
** BPXYPGPS: w_getpsent return data structure
** Used By: GPS
PGPS                                DSECT ,
PGPS                                DS      0F
PGPSSTATUS0                        DS      B      MVS status
PGPSSWAP                            EQU    X'80'    Swapped out
*                                  EQU    X'7F'    Not Used
PGPSSTATUS1                        DS      B      Process status
PGPSSTOPPED                        EQU    X'80'    Stopped process
PGPSTRACE                           EQU    X'40'    PTrace active
PGPSMULTHREAD                      EQU    X'20'    0=One open task in process
PGPSPTHREAD                        EQU    X'10'    0=No pthread task in process
PGPSMULPROCESS                     EQU    X'08'    0=One process in addr space
*                                  EQU    X'07'    Not Used
PGPSSTATUS2                        DS      B      System Call Status
PGPSLENERR                         EQU    X'80'    PGPSLENGTH conflict
*                                  EQU    X'7F'    Not Used
PGPSSTATUS3                        DS      CL1     State of reported task - with
*                                  PGPSPTHREAD=0 the most recent created thread
*                                  PGPSPTHREAD=1 the initial pthread task (IPT)
PGPSMSGRCV                         EQU    C'A'     IPC MSGRCV WAIT
PGPSMSGSD                          EQU    C'B'     IPC MSGSD WAIT
PGPSWAITC                         EQU    C'C'     COMM KERNELWAIT

```

```

PGPSSEMOP      EQU  C'D'    IPC SEMOP WAIT
PGPSFREEZE     EQU  C'E'    QUIESCFREEZE
PGPSWAITF      EQU  C'F'    F S  KERNEL WAIT
PGPSMVSPAUSE   EQU  C'G'    MVSPAUSE
PGPSZOMBIE2    EQU  C'L'    PROCESS TERMINATED AND STILL
*              *           SESSION OR PROCESS GROUP LEADER
PGPSWAITO      EQU  C'K'    OTHER KERNEL WAIT
PGPSQUIESCET    EQU  C'Q'    QUIESCE TEMRINATION WAIT
PGPSRUN        EQU  C'R'    NOT KERNEL WAIT
PGPSSLEEP      EQU  C'S'    SLEEP() ISSUED
PGPSCHILD      EQU  C'W'    WAITING FOR CHILD
PGPSFORK       EQU  C'X'    FORK NEW PROCESS
PGPSZOMBIE     EQU  C'Z'    PROCESS TERMINATED AND PARENT
*              *           HAS NOT ISSUED WAIT SYSCALL
PGPSPID        DS    F      Process ID
PGSPPID        DS    F      Parent ID
PGSSID         DS    F      Session ID (leader)
PGPSGPID       DS    F      Process Group
PGPSFGPID      DS    F      Foreground Process Group
PGPSEUID       DS    F      Effective User ID
PGPSRUID       DS    F      Real User ID
PGPSSUID       DS    F      Saved Set User ID
PGPSEUID       DS    F      Effective Group ID
PGPSRGID       DS    F      Real Group ID
PGPSSGID       DS    F      Saved Set Group ID
PGPSTSIZE      DS    F      Total size
PGPSSTARTTIME  DS    F      Starting time, GMT since EPOCH
PGPSUSERTIME   DS    F      User CPU time (clock_t)
PGPSSYSTIME    DS    F      System CPU time (clock_t)
PGPSCONTTYBLEN DC    A(&VAR1L) L'PGPSCONTTYBUF
PGPSCONTTYPTR  DC    A(PGPSCONTTYBUF) ->PGPSCONTTYBUF
PGPSPATHBLEN   DC    A(&VAR2L)  L'PGPSPATHBUF
PGPSPATHPTR    DC    A(PGPSPATHBUF) ->PGPSPATHBUF
PGPSCMDBLEN    DC    A(&VAR3L)  L'PGPSCMDBUF
PGPSCMDPTR     DC    A(PGPSCMDBUF) ->PGPSCMDBUF
PGPSSERVERTYPE DS    F      Server type (FILE=1, LOCK=2)
PGPSSERVERNAME DS    CL32     Name supplied on registration
PGPSMAXVNODETOKENS DS    F      Max number of VNode Toks allowed
PGPSVNODETOKENCOUNT DS    F      Current number of VNode Tokens
PGPSSERFLAGS   DS    F      Server flags
PGPSSYSCALLCOUNT DS    F      Count of syscalls this process
PGPSJOBNAME     DS    CL8      AscbJBNI/JBNS  JobName
PGPSWAITTIME   DS    F      Since Kern Wait Started
PGPSASID       DS    FL2      Address space ID
                DS    CL6
                SPACE ,
PGPS#LENGTH    EQU  *-PGPS    Length of this structure
* Variable portion - Controlling terminal buffer
*
* Notes on format of controlling terminal string in PGPSCONTTYBUF
* 1. Controlling terminal string returned in PGPSCONTTY is
*    null-terminated.
* 2. The PGPSCONTTYLEN value does NOT include the terminating
*    null character.
PGPSCONTTYBUF   DS    &VAR1A    ConTty Len+Buf
PGPSCONTTYLEN   DS    &VAR1B    Length ConTty returned
PGPSCONTTY      DS    &VAR1C    ConTty (len+1-th char=null)
*
* Notes on format of path string in PGPSPATHBUF:
* 1. Pathname returned in PGPSPATH is null-terminated.
* 2. The PGPSPATHLEN value does NOT include the terminating null
*    character.
* 3. TSO (non-shell) pathnames may be padded with spaces to eight
*    characters.
PGPSPATHBUF     DS    &VAR2A    Pathname Len+Buf
PGPSPATHLEN     DS    &VAR2B    Length Pathname returned
PGPSPATH        DS    &VAR2C    Pathname (len+1-th char=null)
*
* Notes on format of PGPSCMDBUF:
* 1. PGPSCMD consists of one or more character fields representing
*    the command and its arguments (if any).
* 2. Each character field consists of a four byte length field and
*    a null-terminated character string.
* 3. TSO (non-shell) commands may be padded with spaces to eight
*    characters.
* 4. Unlike PGPSCONTTYLEN and PGPSPATHLEN, each character field
*    length value DOES include the null-terminating character.
* 5. The PGPSCMDLEN value is the sum of all character fields (length
*    fields and character strings).
PGPSCMDBUF      DS    &VAR3A    Command Len+Buf
PGPSCMDLEN      DS    &VAR3B    Length Command returned

```

```

PGPSCMD      DS      &VAR3C      Command (array of len, element)
PGPS#STORAGE EQU      *-PGPS      Length, total area used
** BPXYPGPS End

```

```

                BPXYPGPS      VARLEN=(1028,1028,1028)
** BPXYPGPS: w_getpsent return data structure
** Used By: GPS
PGPS           DSECT ,
PGPSSTATUS0    DS      B          MVS status
PGPSSWAP        EQU     X'80'      Swapped out
*              EQU     X'7F'      Not Used
PGPSSTATUS1    DS      B          Process status
PGPSSTOPPED    EQU     X'80'      Stopped process
PGPSTRACE       EQU     X'40'      PTrace active
PGPSMULTHREAD  EQU     X'20'      0=One open task in process
PGPSPTHREAD    EQU     X'10'      0=No pthread task in process
PGPSMULPROCESS EQU     X'08'      0=One process in addr space
*              EQU     X'07'      Not Used
PGPSSTATUS2    DS      B          System Call Status
PGPSLENERR     EQU     X'80'      PGPSLENGTH conflict
*              EQU     X'7F'      Not Used
PGPSSTATUS3    DS      CL1        State of reported task - with
*              PGPSPTHREAD=0 the most recent created thread
*              PGPSPTHREAD=1 the initial pthread task (IPT)
PGPSMSGRCV     EQU     C'A'       IPC MSGRCV WAIT
PGPSMSGSDND    EQU     C'B'       IPC MSGSDND WAIT
PGPSWAITC      EQU     C'C'       COMM KERNELWAIT
PGPSSEMOP      EQU     C'D'       IPC SEMOP WAIT
PGPSFREEZE     EQU     C'E'       QUIESCEFREEZE
PGPSWAITF      EQU     C'F'       F S KERNEL WAIT
PGPSMVSPAUSE   EQU     C'G'       MVSPAUSE
PGPSZOMBIE2    EQU     C'L'       PROCESS TERMINATED AND STILL
*              SESSION OR PROCESS GROUP LEADER
PGPSWAITO      EQU     C'K'       OTHER KERNEL WAIT
PGPSQUIESCET   EQU     C'Q'       QUIESCE TEMRINATION WAIT
PGPSRUN        EQU     C'R'       NOT KERNEL WAIT
PGPSSLEEP      EQU     C'S'       SLEEP() ISSUED
PGPSCCHILD     EQU     C'W'       WAITING FOR CHILD
PGPSFORK       EQU     C'X'       FORK NEW PROCESS
PGPSZOMBIE     EQU     C'Z'       PROCESS TERMINATED AND PARENT
*              HAS NOT ISSUED WAIT SYSCALL
PGPSPID        DS      F          Process ID
PGPSPPID       DS      F          Parent ID
PGPSSID        DS      F          Session ID (leader)
PGPSPGPID      DS      F          Process Group
PGPSFGPID      DS      F          Foreground Process Group
PGPSEUID       DS      F          Effective User ID
PGPSRUID       DS      F          Real User ID
PGPSSUID       DS      F          Saved Set User ID
PGPSEUID       DS      F          Effective Group ID
PGPSRGID       DS      F          Real Group ID
PGPSSGID       DS      F          Saved Set Group ID
PGPSTSIZE      DS      F          Total size
PGPSSTARTTIME  DS      F          Starting time, GMT since EPOCH
PGPSUSERTIME   DS      F          User CPU time (clock_t)
PGPSSYSTIME    DS      F          System CPU time (clock_t)
PGPSCONTTYBLEN DC      A(1028)    L'PGPSCONTTYBUF
PGPSCONTTYPTR  DC      A(PGPSCONTTYBUF) ->PGPSCONTTYBUF
PGPSPATHBLEN   DC      A(1028)    L'PGPSPATHBUF
PGPSPATHPTR    DC      A(PGPSPATHBUF) ->PGPSPATHBUF
PGPSCMDBLEN    DC      A(1028)    L'PGPSCMDBUF
PGPSCMDPTR     DC      A(PGPSCMDBUF) ->PGPSCMDBUF
PGPSSERVERTYPE DS      F          Server type (FILE=1, LOCK=2)
PGPSSERVERNAME DS      CL32       Name supplied on registration
PGPSMAXVNODETOKENS DS      F      Max number of VNode Toks allowed
PGPSVNODETOKENCOUNT DS      F    Current number of VNode Tokens
PGPSSERFLAGS   DS      F          Server flags
PGPSSYSCALLCOUNT DS      F      Count of syscalls this process
PGPSJOBNAME    DS      CL8        AscbJBNI/JBNS JobName
PGPSWAITTIME   DS      F          Since Kern Wait Started
PGPSASID       DS      FL2        Address space ID
*              DS      CL6
PGPSSTARTTIME64 DS      D          Start time, since Epoch
PGPS#LENGTH    EQU     *-PGPS     Length of this structure
* Variable portion - Controlling terminal buffer
*
* Notes on format of controlling terminal string in PGPSCONTTYBUF
* 1. Controlling terminal string returned in PGPSCONTTY is
*    null-terminated.
* 2. The PGPSCONTTYLEN value does NOT include the terminating

```

```

* null character.
PGPSCONTTYBUF      DS      0CL1028   ConTty Len+Buf
PGPSCONTTYLEN      DS      FL4       Length ConTty returned
PGPSCONTTY         DS      CL1024    ConTty (len+1-th char=null)
*
* Notes on format of path string in PGPSPATHBUF:
* 1. Pathname returned in PGPSPATH is null-terminated.
* 2. The PGPSPATHLEN value does NOT include the terminating null
* character.
* 3. TSO (non-shell) pathnames may be padded with spaces to eight
* characters.
PGPSPATHBUF        DS      0CL1028   Pathname Len+Buf
PGPSPATHLEN        DS      FL4       Length Pathname returned
PGPSPATH           DS      CL1024    Pathname (len+1-th char=null)
*
* Notes on format of PGPSCMDBUF:
* 1. PGPSCMD consists of one or more character fields representing
* the command and its arguments (if any).
* 2. Each character field consists of a four byte length field and
* a null-terminated character string.
* 3. TSO (non-shell) commands may be padded with spaces to eight
* characters.
* 4. Unlike PGPSCONTTYLEN and PGPSPATHLEN, each character field
* length value DOES include the null-terminating character.
* 5. The PGPSCMDLEN value is the sum of all character fields (length
* fields and character strings).
PGPSCMDBUF         DS      0CL1028   Command Len+Buf
PGPSCMDLEN         DS      FL4       Length Command returned
PGPSCMD            DS      CL1024    Command (array of len, element)
PGPS#STORAGE       EQU      *-PGPS   Length, total area used
** BPXYPGPS End

```

## BPXYPGTH — Map the \_\_getthent input/output structure

```

** BPXYPGTH: __getthent input and output structures
** Used By: GTH
PGTHA              DSECT ,          I N P U T - - - - -
PGTHACONTINUE      DS      0CL14
PGTHAID            DS      F        PROCESS ID (IGNORED IF FIRST)
PGTHATHID          DS      CL8      THREAD ID (IGNORED IF FIRST/LAST)
PGTHAACCESSPID     DS      FL1      FIRST, CURRENT, NEXT
PGTH#NEXT          EQU      2       NEXT AFTER SPECIFIED
PGTH#CURRENT       EQU      1       AS SPECIFIED
PGTH#FIRST         EQU      0       FIRST (EQUIV NEXT WITH PID=0)
PGTH#LAST          EQU      3       only with PGTHIACCESSTHID
PGTHAACCESSTHID    DS      FL1      FIRST, CURRENT, NEXT, LAST
* ONLY FLAG1 BITS THREAD AND PTAG WILL BE CONSIDERED WHEN
* ACCESSPID=CURRENT AND ACCESSTHID=NEXT
PGTHAACSID         DS      FL2      FILTER - ASID
* LOGINNAME COMPARISON WILL LOOK FOR UNIX ALIAS. IF PGHTALOGINNAME
* IS NOT AN ALIAS, IT WILL BE SHIFTED TO UPPER CASE AND CHECKED
* AGAINST MVS ID.
PGTHALOGINNAME     DS      CL8      FILTER - USERID ALIAS OR MVS
PGTHAFLAG1         DS      FL1      WHAT OUTPUT AREAS TO INCLUDE
PGTHAPROCESS       EQU      X'80'   PGTHC, PROCESS DATA
PGTHACONTTY        EQU      X'40'   PGTHD, CONTTY
PGTHAPATH          EQU      X'20'   PGTHE, PATH
PGTHACOMMAND       EQU      X'10'   PGTHF, CMD & ARGS - UP TO
*                                     1024 BYTES
PGTHAFILEDATA      EQU      X'08'   PGTHG, FILE DATA
PGTHATHREAD        EQU      X'04'   PGTHJ, THREAD DATA
PGTHAPTAG          EQU      X'02'   PGTHK, PTAG (NEEDS PGTHJ)
PGTHACOMMANDLONG   EQU      X'01'   PGTHF, CMD & ARGS - UP TO
*                                     2048 BYTES
PGTHAFLAG1B2       DS      FL1      Flag1, Byte 2
PGTHATHREADFAST    EQU      x'80'   SKIP SCHEDULING SRB
PGTHAFILEPATH      EQU      x'40'   RETURN PATH NAME
PGTHATHDSIGMASK    EQU      x'20'   RETURN SIGMASK IN PGTHJ
PGTHANAMESPACE     EQU      x'10'   RETURN NAMESPACE DATA @D9A
PGTHAABSPATH       EQU      x'08'   RETURN ABSPATH IN PGTHE @DAA
PGTHAPROCSIGPND    EQU      x'04'   RETURN SIGPND IN PGTHC @P8A
PGTHASIGNALS       EQU      x'02'   RETURN PGTHM WITH PGTHJ @P8A
PGTHA#LEN          EQU      *-PGTHA

```

```

SPACE ,
PGTHB      DSECT ,      O U T P U T - - - - -
PGTHBID    DS      CL4   "gthb"
PGTHBCONTINUE DS      0CL14 NEXT VALUE FOR PGTHACONTINUE
PGTHBPID   DS      F     PROCESS ID
PGTHBTHID  DS      CL8   THREAD ID
PGTHBACCESSPID DS      FL1 CURRENT/FIRST/NEXT
PGTHBACCESSTHID DS      FL1 CURRENT/FIRST/NEXT/LAST
DS      FL2
PGTHBLENUSED DS      F     LENGTH OF OUTPUT BUFFER USED
PGTHBLIMITC DS      CL1   Can be 0, N, A, V
PGTHBOFFC   DS      FL3   OFFSET OF PROCESS AREA
*
PGTHBLIMITD DS      CL1   Can be 0, N, A, V, S, X
PGTHBOFFD   DS      FL3   OFFSET OF CONTTY AREA
*
PGTHBLIMITE DS      CL1   Can be 0, N, A, V, S, X
PGTHBOFFE   DS      FL3   OFFSET OF PATH AREA
*
PGTHBLIMITF DS      CL1   Can be 0, N, V, A, S, X
PGTHBOFFF   DS      FL3   OFFSET OF COMMAND AREA
*
PGTHBLIMITG DS      CL1   Can be 0, N, A, V, X
PGTHBOFFG   DS      FL3   OFFSET OF FILE DATA AREA
*
PGTHBLIMITJ DS      CL1   Can be 0, N, A, V, X
PGTHBOFFJ   DS      FL3   OFFSET OF THREAD AREA
*
PGTHBLIMITL DS      CL1   Can be 0, N, A, V, S      @D9A
PGTHBOFFL   DS      FL3   OFFSET OF NAMESPACE AREA @D9A
*
PGTHB#LEN   EQU      *-PGTHB
SPACE ,
*****
*
* Values for PGTHBLIMITx fields and PgthGLimitH, PgthJLimitJ, and
* PgthJLimitK fields
*
* Values for PGTHBLIMITx fields are
*   N - associated area was not requested to be filled in
*   A - the section was completely filled in
*   S - the output buffer is not big enough for the requested
*       data. The section has been filled in as much
*       as possible.
*   V - the section was started but could not be completed
*       due to a system error. Data in this section can
*       not be trusted.
*   X - the requested data was not available
*   0 - processing did not get far enough to fill out this
*       section of the buffer. Most likely, a buffer full
*       condition occurred while filling out a previous section
*       and the service stops further processing and returns
*       EINVAL JrBuffTooSmall to the caller.
*
*****
PGTH#NOTREQUESTED EQU      C'N'   Associated PgthA.. bit off
PGTH#OK           EQU      C'A'   All data included
PGTH#STORAGE      EQU      C'S'   output buffer exhausted
* EXHAUSTED STORAGE < 1ST PGTHJ RESULTS IN -1 EINVALID JRBUFFTTOOSMALL
PGTH#VAGUE        EQU      C'V'   Changed out from under us
PGTH#NOTCONNECTED EQU      C'X'   Need data not connected
SPACE ,
* USING PGTHC,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFC
PGTHC      DSECT ,      P R O C E S S - - - - -
PGTHCID    DS      CL4   "gthc"
PGTHCFLAG1 DS      FL1
PGTHCMULPROCESS EQU      X'80'   HAD OR HAS MULTIPLE PROCESSES
PGTHCSWAP   EQU      X'40'   TCBOUT
PGTHCTRACE  EQU      X'20'   THREAD IS BEING TRACED
PGTHCSTOPPED EQU      X'10'   STOPPED
PGTHCINCOMPLETE EQU      X'08'   NOT ALL BLOCKS PRESENT
PGTHCZOMBIE EQU      X'04'   PROCESS IS A ZOMBIE
PGTHCBLOCKING EQU      X'02'   Shutdown blocking
PGTHCPERM   EQU      X'01'   SHUTDOWN permanent
PGTHCFLAG2  DS      FL1
PGTHCMEMLTYPE EQU      X'80'   ON - MemLimit is a BinMult
PGTHCRESPAWN EQU      X'40'   respawnable process
PGTHCUSERTRACEACT EQU      X'20'   User Syscall Trace Active
PGTHCCONTAINER EQU      X'10'   Process is in a container @DBA
PGTHCFLAG3  DS      FL1
PGTHCMEMUTYPE EQU      X'80'   ON - MemUsage is a BinMult
DS      1FL1

```

```

PGTHCPID      DS    F    PROCESS ID
PGTHCPPID      DS    F    PARENT ID
PGTHCPGPID      DS    F    PROCESS GROUP
PGTHCSID      DS    F    SESSION ID
PGTHCFGPID      DS    F    FOREGROUND PROCESS GROUP
PGTHCEUID      DS    F    EFFECTIVE USER ID
PGTHCRUID      DS    F    REAL USER ID
PGTHCSUID      DS    F    SAVED SET USER ID
PGTHCEGID      DS    F    EFFECTIVE GROUP ID
PGTHCRGID      DS    F    REAL GROUP ID
PGTHCSGID      DS    F    SAVED SET GROUP ID
PGTHCTSIZE      DS    F    Allocated user storage > 16 MB
PGTHCSYSCALLCOUNT DS    F    COUNT OF SLOW-PATH SYSCALLS
*
* PgthCUserTime and PcthCSysTime units are 1/100ths of a second .01
PGTHCUSERTIME    DS    F    TIME SPENT IN USER CODE
PGTHCSYSTIME      DS    F    TIME SPENT IN SYSTEM CODE
PGTHCSTARTTIME    DS    F    TIME PROCESS WAS DUBBED
PGTHCCNTOE      DS    FL2    NO. OE THREADS
PGTHCCNTPTCREATED DS    FL2    NO. PTHREAD CREATED THREADS
PGTHCCNTTTHREADS DS    FL2    COUNT OF ALL THREADS
PGTHCASID        DS    FL2    ADDRESS SPACE ID
PGTHCJOBNAME      DS    CL8    MVS JOB NAME
PGTHCLOGINNAME    DS    CL8    LOGIN NAME - ALIAS OR MVS
PGTHCMEMLIMIT     DS    1FL4   maximum Memlimit in bytes
    ORG    PGTHCMEMLIMIT
PGTHCMEMLIMITVAL DS    1FL3    Hex value
PGTHCMEMMULT      DS    1CL1    multiplier when PGTHCMEMLTYPE
PGTHCMEMUSAGE     DS    1FL4    bytes in use above the 2G bar
    ORG    PGTHCMEMUSAGE
PGTHCMEMUSAGEVAL  DS    1FL3    Hex value
PGTHCMEMMULT      DS    1CL1    multiplier when PGTHCMEMUTYPE
PgthCUserTimeMics DS    XL8    User CPU time in Mics      @D7A
PgthCSysTimeMics  DS    XL8    System CPU time in Mics    @D7A
PGTHCSTARTTIME64  DS    XL8    Time process was dubbed    @D8A
PGTHCPriority     DS    F      Priority value             @DAA
PGTHCSigPnd       DS    XL8    Process pending signal mask @P8A
PGTHCX    DS    0C
*
* *****
*
* PGTHCMEMLIMIT constants are used by PGTHCMEMMULT and
* PGTHCMEMMULT when the TYPE is a binmult.
*
* When PGTHCMEMLTYPE is on PGTHCMEMLIMIT consists of a
* 24bit binary value in the first three bytes followed by
* and ebcdic constant that indicates the denomination.
*
* When PGTHCMEMLTYPE is off PGTHCMEMLIMIT consists of a
* 32bit binary value.
*
* *****
*
PGTH#KILO EQU C'K'      Kilobytes
PGTH#MEGA EQU C'M'      Megabytes
PGTH#GIGA EQU C'G'      Gigabytes
PGTH#TERA EQU C'T'      Terabytes
PGTH#PETA EQU C'P'      Petabytes
PGTHC#LEN EQU *-PGTHC
SPACE ,
* USING PGTHD,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFD
PGTHD      DSECT ,      C O N T T Y - - - - -
PGTHDID     DS    CL4    "gthd"
PGTHDLEN    DS    FL2    Length of ContTy
PGTHDCONTTY DS    CL1024 1024 = max ContTy
SPACE ,
* USING PGTHE,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFE
PGTHE      DSECT ,      P A T H - - - - -
PGTHEID     DS    CL4    "gthe"
PGTHELEN    DS    FL2    Length of Path
PGTHEPATH   DS    CL1024 1024 = max path
SPACE ,
* USING PGTHF,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFF
PGTHF      DSECT ,      C O M M A N D - - - - -
PGTHFID     DS    CL4    "gthf"
PGTHFLEN    DS    FL2    Length of command and arguments
PGTHFCOMMAND DS    CL1024 1024 = max command
    ORG    PGTHFCOMMAND
PGTHFCOMMANDL DS    CL2048 Allow up to 2K for cmd/args @D5A
SPACE ,
* USING PGTHG,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFG

```

```

PGTHG      DSECT ,      F I L E   H E A D E R - - - - -
PGTHGID    DS      CL4      "gthg"
PGTHGLIMITH DS      CL1      Can be 0, V, S, A
PGTHGOFFH  DS      FL3      Offset of PgthH
PGTHGCOUNT DS      F        Count of PgthH elements
PGTHGMAXVNODETOKENS DS      F        MAX NUMBER VNODE TOKENS
PGTHGVNODETOKENCOUNT DS      F        CURRENT NUMBER VNODE TOKENS
PGTHGSERVERFLAGS DS      F        Internal use
PGTHGSERVERNAME DS      CL32     NRegsName from v_reg
PGTHGACTIVEFILES DS      F        Current num of files
PGTHGMAXFILES DS      F        Max num of files allowed
PGTHGSERVERTYPE DS      F        NRegsType from v_reg
PGTHG#LEN  EQU      *-PGTHG
          SPACE ,
PGTHGARRAY DS      0C        first PGTHH
          SPACE ,
* USING PGTHH,Rx where Rx = ADDRESS of PGTHB + PGTHGOFFH
* Increment Rx by PGTHH#LEN until PGTHGCOUNT exhausted
* Increment Rx again by PGTHH#LEN+PGTHHPATHLEN if path data requested
* and 2nd character of PGTHHID is PGTHH#IDP
PGTHH      DSECT ,      F I L E   D A T A - - - - -
PGTHHID    DS      CL2
PGTHH#IDR  EQU      C'rd'     root directory (first)
PGTHH#IDC  EQU      C'cd'     current directory (second)
PGTHH#IDF  EQU      C'fd'     file directory
PGTHH#IDV  EQU      C'vd'     vnode directory
PGTHH#IDP  EQU      C'p'      path included in entry
PGTHHTYPE  DS      BL1      Mapped in BPXYFTYP see FT_DIR +
PGTHHOPEN  DS      BL1      Mapped in BPXYOPNF see O_FLAGS4
PGTHHINODE DS      F        I-NODE see stat()
PGTHHDEVNO DS      F        DEVICE NUMBER see stat()
PGTHH#LEN  EQU      *-PGTHH
          SPACE ,
* USING PGTHHX,Rx where Rx = ADDRESS of PGTHH + PGTHH#LEN 9@D6A
PGTHHX     DSECT ,      F I L E   P A T H   D A T A - -
PGTHHXFLAGS DS      CL1
PGTHHPATHTRUNC EQU      X'80'   partial path returned
PGTHHXRSVD DS      CL1
PGTHHPATHLEN DS      FL2
PGTHHX#LEN EQU      *-PGTHHX
PGTHHPATH  DS      0CL1
          SPACE ,
* USING PGTHJ,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFJ
* Reset Rx to be PGTHB + PGTHJOFFJ for the next thread
PGTHJ      DSECT ,      T H R E A D - - - - -
PGTHJID    DS      CL4      "gthj"
PGTHJLIMITJ DS      CL1      Can be 0, A, S, X
PGTHJOFFJ  DS      FL3      Offset of next PgthJ
PGTHJLIMITK DS      CL1      Can be 0, N, A, S, X
PGTHJOFFK  DS      FL3      Offset of PgthK, this thread
PGTHJTHID  DS      CL8      THREAD ID
PGTHJSYSCALL DS      CL4     SYSCALL (eg. "1FRK" for fork)
PGTHJTCTB  DS      A        TCB ADDRESS
PGTHJTTIME DS      F        TIME RUNNING .001 SECS (millisecs)
PGTHJWTIME DS      F        OE WAITING TIME .001 SECS (milli)
PGTHJLIMITM DS      CL1      Can be 0, N, A, S @P8A
PGTHJOFFM  DS      FL3      Offset of PgthM, this thread @P8A
PGTHJSEMNUM DS      H        SEMAPHORE NUMBER IF STATUS2=D
PGTHJSEMVAL DS      H        SEMAPHORE VALUE IF STATUS2=D
PGTHJLATCHWAITPID DS      F        LATCH PROCESS ID WAITED FOR
PGTHJPENMASK DS      XL8     SIGNAL PENDING MASK
PGTHJLOGINNAME DS      CL8    LOGIN NAME - ALIAS or MVS
PGTHJPREVSC DS      5CL4     LAST FIVE SYSCALLS
          ORG      PGTHJPREVSC
          DS      CL12
PGTHJSIGMASK DS XL8
*
* End of PGTHJPREVSC/PGTHJSIGMASK UNION
*
PGTHJSTATUSCHARS DS      0CL5     STATUS
          SPACE ,
PGTHJSTATUS1 DS      CL1      STATUS 1
PGTHJ#PTHDCREATED EQU      C'J'   pthread created
          SPACE ,
PGTHJSTATUS2 DS      CL1      STATUS 2
PGTHJ#MSGRCV EQU      C'A'       msgrcv wait
PGTHJ#MSGSEND EQU      C'B'       msgsnd wait
PGTHJ#WAITC EQU      C'C'       communication wait
PGTHJ#SEMOP EQU      C'D'       see PgthJSemVal/SemNum
PGTHJ#WAITF EQU      C'F'       file system wait
PGTHJ#MVSPAUSE EQU      C'G'       MVS in pause
PGTHJ#WAITO EQU      C'K'       other kernel wait

```



```

PGTHJ#WAITP      EQU   C'P'      PTwaiting
PGTHJ#RUN        EQU   C'R'      running / non-kernel wait
PGTHJ#SLEEP      EQU   C'S'      sleep
PGTHJ#CHILD      EQU   C'W'      waiting for child
PGTHJ#FORK       EQU   C'X'      fork new process
PGTHJ#MVSWAIT    EQU   C'Y'      MVS wait
                SPACE ,
PGTHJSTATUS3     DS     CL1      STATUS 3
PGTHJ#MEDIUMWGHT EQU   C'N'      medium weight thread
PGTHJ#ASYNC      EQU   C'O'      asynchronous thread
PGTHJ#IPT        EQU   C'U'      Initial process thread
PGTHJ#ZOMBIE     EQU   C'Z'      Process terminated and parent
*                                     has not completed wait
                SPACE ,
PGTHJSTATUS4     DS     CL1      STATUS 4
PGTHJ#DETACHED   EQU   C'V'      thread is detached
                SPACE ,
PGTHJSTATUS5     DS     CL1      STATUS 5
PGTHJ#FREEZE     EQU   C'E'      quiesce freeze
                DS     CL3
PgthJTTIMEmics   DS     XL8      TCBTime in microseconds    @D7A
PgthJWTIMEmics   DS     XL8      Wait time in microseconds  @D7A
                SPACE ,
PGTHJ#LEN        EQU   *-PGTHJ
* USING PGTHH,Rx where Rx = ADDRESS of PGTHB + PGTHJOFFK
PGTHK            DSECT ,        P T A G - - - - -
PGTHKDATALEN     DS     F        LENGTH TO TRAILING NULL
PGTHKDATA        DS     CL68     SEE pthread_tag_np
PGTHK#LEN        EQU   *-PGTHH
                SPACE ,
* USING PGTHL,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFL    @D9A
* Reset Rx to be PGTHB + PGTHLOFFL for the next namespace @D9A
PGTHL            DSECT ,        N A M E S P A C E - - - - - @D9A
PGTHLID          DS     CL4      "gthl"                    @D9A
PGTHLLIMITL      DS     CL1      Can be 0, A, V, S          @D9A
PGTHLOFFL        DS     FL3      Offset of next PgthL       @D9A
PGTHLNSID        DS     CL8      Namespace ID               @D9A
PGTHLNSTYPE      DS     CL1      Namespace type             @D9A
PGTHL#PID        EQU   C'P'      PID namespace             @D9A
PGTHL#IPC        EQU   C'I'      IPC namespace              @D9A
PGTHL#MOUNT      EQU   C'M'      Mount namespace            @D9A
PGTHL#NETWORK    EQU   C'N'      Network namespace          @D9A
PGTHL#UTS        EQU   C'U'      UTS namespace              @D9A
PGTHL#USER       EQU   C'R'      User namespace             @D9A
                SPACE ,
PGTHL#LEN        EQU   *-PGTHL    @D9A
* USING PGTHM,Rx where Rx = ADDRESS of PGTHB + PGTHJOFFM    @P8A
PGTHM            DSECT ,        S I G N A L S - - - - - @P8A
PGTHMSIGIGN      DS     XL8      Ignored signals mask        @P8A
PGTHMSIGCGT      DS     XL8      Caught signals mask         @P8A
PGTHMSIGBLK      DS     XL8      Blocked signals mask        @P8A
PGTHM#LEN        EQU   *-PGTHM    @P8A
                SPACE ,
** BPXYPGTH End

```

## BPXYPOE — Map poe syscall parameters

This structure is passed to the poe syscall.

```

                BPXYPOE ,
POE            DSECT
POEOPTIONS DS 1FL4 +00 Options for POE
POENTRYTYPE DS 1FL4 +04 Point Of Entry Type
POENTRYLEN DS 1FL4 +08 Point Of Entry Length
                DS 1CL0004 +0C Reserved
POENTRYPTR64 DS 0CL0008 +10 64 Address of Port of Entry
                DS 1CL0004 +10 Padding
POENTRYPTR DS 1AL4 +14 Address of Point Of Entry
POE#LEN EQU 24 Length of POE
POE#ENTRYSOCKET EQU 1 Entry is a file descriptor for a socket file
POE#ENTRYSOCKETLEN EQU 4 Length of file descriptor of a socket file
POE#ENTRYFILE EQU 2 Entry is a file descriptor for a non-socket X
                    file
POE#ENTRYFILELEN EQU 4 Length of file descriptor for a non-socket X
                    file
*
* Options for Poe
*

```

```

*
POE#SCOPETHREAD EQU 1   Thread scope
POE#SCOPEPROCESS EQU 2   Process scope
POE_LEN EQU *-POE

```

## BPXPOLL – Map poll syscall parameters

This structure is passed to the poll syscall.

```

                BPXPOLL
** BPXPOLL: POLLFD structure for poll syscall
** Used By: POL
POLLFD          DSECT
POLLHFD         DS      FL4      File descriptor
                ORG      POLLHFD
POLLHMQID       DS      FL4      Message queue identifier
POLLEVENTS      DS      0XL2     Events
                DC      XL1'0'    Reserved
                DS      XL1      POLLEVENTS+1
POLLEPRI        EQU      X'10'   High-pri data may be recv'd
POLLEWRBAND     EQU      X'08'   Priority data may be written
POLLEWRNORM     EQU      X'04'   Data on band 0 may be written.
POLLEOUT        EQU      X'04'   Same as WrNorm
POLLEIN         EQU      X'03'   Same as RdNorm
POLLERDBAND     EQU      X'02'   Non-0 band data may be read
POLLERDNORM     EQU      X'01'   Data on band 0 may be read.
POLLREVENTS     DS      0XL2     Returned events
                DS      XL1      Reserved
                DS      XL1      POLLREVENTS+1
POLLRINVAL     EQU      X'80'   Invalid FD member.(Revent Only)
POLLRHUP        EQU      X'40'   Hangup occurred (Revent Only)
POLLRERR        EQU      X'20'   Error occurred. (Revent Only)
POLLRPRI        EQU      X'10'   High-pri data may be recv'd
POLLRWRBAND     EQU      X'08'   Priority data may be written
POLLRWRNORM     EQU      X'04'   Data on band 0 may be written.
POLLROUT        EQU      X'04'   Same as WrNorm
POLLRI          EQU      X'03'   Same as RdNorm
POLLRRDBAND     EQU      X'02'   Non-0 band data may be read
POLLRRDNORM     EQU      X'01'   Data on band 0 may be read.
POLLFD#LENGTH  EQU      *-POLLFD
*
#POLLEMASK      EQU      X'001F'
#POLLRDMASK     EQU      X'00130000' All Read bits
#POLLWRMASK     EQU      X'000C0000' All Write bits
#POLLPRIMASK    EQU      X'00100000' The PollPri bit
#POLLINMASK     EQU      X'00030000' Pollin rdnorm rdband bits
#POLLRNMASK     EQU      X'00010000' Read Normal
#POLLWNMASK     EQU      X'00040000' Write Normal
#POLLEVMAK      EQU      X'001F0000' Events
*
*
** BPXPOLL End

```

## BPXYPPSD – Map signal delivery data

This structure is passed to a signal interface routine (SIR). AMODE 64 callers use “BPXYPPSD – Map signal delivery data” on page 1206.

```

                BPXYPPSD
** BPXYPPSD: Signal Data Area
** Used By: User written signal interrupt routines
PPSD            DSECT
PPSID           DC      C'PPSD'   Eye catcher
PPSD#ID         EQU      C'PPSD'   Control Block Acronym
PPSDSP          DS      FL1       Subpool number of this PPSD
PPSD#SP         EQU      230      Subpool for the PPSD
PPSDLEN         DC      AL3(PPSD#LENGTH) Length this structure
*
* *****
* PpsdSIRParms is used to setup up a parameter list to the
* Signal Interface Routine (SIR). When the SIR is invoked, the
* address of PpsdSIRParms field is set in Register 1. The

```

```

*      PpsdAddrPpsd contains the address of the Ppsd.
*      *****
*
PPSDSIRPARMS      DS      0A      SIR Parameters
PPSDADDRPPSD      DC      A(PPSD) Pointer to the top of the Ppsd
PPSDSIRPARMEND    EQU     X'80'   End of Parameters flag set on
PPSDTRMEXITSTATUS DS      F      4 Byte status passed to PRTRM
PPSDSIGNUM        DS      F      Signal number
PPSDFL            DS      XL2     X'7FFF' reserved
                ORG      PPSDFL
PPSDFLAGS2A       DS      0B
PPSDQUIESCEFREEZE EQU     X'80'   Interrupt due to freeze
PPSDSIRCOMPLETE   EQU     X'40'   Sir done with async I/O exits
PPSDPROCDFLT      EQU     X'20'   Process default
PPSDSIGQUEUE      EQU     X'10'   NSSGQ queued signal
PPSDREDRIVE       EQU     X'08'   SPB will Resend signal later
PPSDJUMPBACK      EQU     X'04'   SPB return to point of interrupt
PPSDMASKONLY      EQU     X'02'   SPB restore mask only
PPSDSIGHSTOP      EQU     X'01'   Interrupt due to thread-stop
*
                ORG      PPSDFL+0001
PPSDFLAGS2B       DS      B
PPSDQUIESCEANDGET EQU     X'80'   Interrupt due to
*                               pthread_quiesce_and_get_np
PPSDF2_64         EQU     X'40'   Use PSWxxx64 fields
PPSDACTION        DS      B      Action for this signal
*                               catch
*                               SIR determines default action
PPSDFLAGS         DS      B      X'00' reserved
PPSDASYNC         EQU     X'80'   Signal delivered Asynchronously
PPSDDUMP          EQU     X'40'   Dump for terminating signals
PPSDPTHREADKILL   EQU     X'20'   Signal sent via BPX1PTK
PPSDTHISTHREADGEN EQU     X'10'   Sending=Receiving thread
PPSDSIGNAL        EQU     X'08'   Interrupt due to signal
PPSDCANCEL        EQU     X'04'   Interrupt due to cancel
PPSDQUIESCE       EQU     X'02'   Interrupt due to quiesce
PPSDIPT           EQU     X'01'   If ON then this is the IPT
PPSDSAHANDLER     DS      A      Addr of catcher function
PPSDSAMASK        DS      XL8    Signal mask set by BPX1SIA for
*                               this signal
PPSDSAFLAGS       DS      XL4    X'00FFFFFF' reserved
PPSDNOCLDSTOP     EQU     X'80'   Do not generate SIGCHLD on stops
PPSDOLDSTYLE      EQU     X'40'   Signal defined by signal() funct.
PPSDONSTACK       EQU     X'20'   Deliver on alternate stack
PPSDRESETHAND     EQU     X'10'   Reset action on delivery
PPSDRESTART       EQU     X'08'   Restart interruptable funcs
PPSDSIGINF        EQU     X'04'   Pass sig info to catcher
PPSDNOCLDWAIT     EQU     X'02'   Don't create zombie on exit
PPSDNODEFER       EQU     X'01'   Don't block sig on delivery
PPSDCURRENTMASK   DS      XL8    This is the signal mask to be set
*                               when the signal catcher returns.
*                               Signal mask at time of interrupt
*                               except for sigsuspend case. If
*                               signal during sigsuspend, then
*                               this mask is the signal mask prior
*                               to call to sigsuspend.
PPSDSIR           DS      A      Addr Signal interrupt routine
PPSDUSERDATA      DS      A      User data speicified on BPX1MSS
PPSDGENREGS       DS      CL64   Users general regs at interrupt
PPSDPSW          DS      XL8     Users PSW at interrupt
PPSDARREGS        DS      16F    Users AR regs at interrupt
PPSDKILDATA       DS      FL2    User specified data on BPX1KIL
PPSDKILOPTS       DS      XL2    X'7FFF' reserved
*                               User specified options on BPX1KIL
PPSDPTBYPASS      EQU     X'80'   Ptrace Bypass option in effect
PPSDKERNSICODE    EQU     X'40'   PpsdKilData=Kern set SiCode
PPSDAPPLSICODE    EQU     X'20'   PpsdKilData=Appl set SiCode
PPSDCONSCANCEL    EQU     X'10'   Console MODIFY cancel qualifier
*                               in PpsdKilData
PPSDSUPERKILL     EQU     X'08'   Superkill option on BPX1KIL
PPSDTRACEOVERRIDE EQU     X'04'   SYSCALL Trace Override Option
PPSDTRACEACTION   EQU     X'02'   SYSCALL Trace Action Setting

PPSDQUIESCEDATA   DS      F      Quiesce_Data specified on BPX1QUT
PPSDLASTPTSIG     DS      F      Last Ptraced Signal
PPSDSIGACTIONDATA DS      F      User_Data specified on BPX1SIA
PPSDPTXLWAPTR     DS      A      Threads workarea address specified
*                               on BPX1PTC (pthread_create). This
*                               address is zero if the thread was
*                               not pt_created.
PPSDSENDINGTHREAD DS      CL8    Sending thread id
PPSDTARGETTHREAD  DS      CL8    Target thread id

```

PPSDSENDINGPID	DS	F	Sending process id
PPSDSENDINGUID	DS	F	Sending real uid
PPSDSIADDR	DS	A	Address of faulting instruction
*			for SIGILL, SIGFPE, SIGSEGV
PPSDSISTATUS	DS	F	Exit status or signal
PPSDSIBAND	DS	F	Band event
PPSDERRNO	DS	F	Error return code
PPSDCATCHERMASK	DS	XL8	Signal Mask to be set before signal
*			catcher is called. If signal during
*			sigsuspend then this field is same
*			as mask specified on sigsuspend. If
*			not sigsuspend, then PpsdCatcherMask
*			and PpsdCurrentMask are equal.
PPSDRES10	DS	25F	Reserved
PPSDSQV	DS	F	Signal si_value
PPSDREDRIVETIME	DS	F	Time to delay signal 1000 per mic
PPSDG64H	DS	16F	Users G64H at interrupt
PPSDRRTRMSGTHID	DS	CL8	Sending thread id for MSG
*			BPXP0101
PPSDSENDINGJOBNAME	DS	CL8	Jobname of thread sending signal
*			
	DS	4F	Reserved
PPSDRES11	DS	22F	Reserved in 31 bit mode
	DS	4F	Reserved
	DS	FL2	Reserved
PPSDAIOCB64	DS	FL2	Amode(64) Exit Flags
PPSDEXCOUNT	DS	FL2	Count of PpsdAiocb's
PPSDEXLASTIX	DS	FL2	Last array index used
PPSDAIOCB	DS	12D	Aiocb Array for Async Exit
PPSDEND	DS	0D	End of PPSD on double word
PPSD#LENGTH	EQU	*-PPSD	Length of this structure
** BPXYPPSD End			

## BPXYPRLI — Process-level information

```

                BPXYPRLI ,
** BPXYPRLI: Process Level Information
PRLI           DSECT ,
PRLIID         DC      C'PRLI'   EBCDIC ID
PRLISP         DS      FL1       Subpool number of this Prli
PRLILEN        DS      FL3       Length of this Prli
PRLIPROCESSID  DS      F         Process ID. Used for fast getpid()
PRLICATCHERMASK DS     BL8       Mask of signals that may be caught
PRLIOAPB       DS      A         Oapb Addr of this process
PRLIFLAG       DS      BL1       Flag byte
PRLIF1MED      EQU     B'10000000' Process is medium weight local
PRLIF1DISSIG   EQU     B'01000000' Disable signal delivery
PRLISYSCONFOK  EQU     B'00100000' SC_fields valid. Note, that      X
                                                this implies the OEXT SC_    X
                                                fields are also valid
PRLIF1TERMT    EQU     B'00010000' Terminate threads
PRLIMAGICNUMBER DS     CL2       Magic Number Characters
                DS      CL1       Reserved
PRLIL16JRC     DS      F         Return code area for L16J FastCGI
PRLIRUID       DS      A         Real Uid addr
PRLIEUID       DS      A         Effective Uid addr
PRLIRGID       DS      A         Real Gid addr
PRLIEGID       DS      A         Effective gid
PRLIPROCGRPID  DS      A         Process Group ID addr
PRLIPARENTPID  DS      A         Parent Process ID addr
PRLITHREADTASKSMAX DS     A         SC_THREAD_TASK_MAX_NP value addr
PRLITHREADSMAX DS      A         SC_THREADS_MAX_NP value address
PRLICHILDMAX   DS      A         SC_CHILD_MAX value addr
PRLIOPENMAX    DS      A         SC_OPEN_MAX value address
PRLIMAPMEMMAX  DS      A         SC_MMAP_MEM_MAX value address
PRLIEND        DS      0C        End of Prli
PRLI#ID        EQU     C'PRLI'   Control Block Acronym
PRLI#LEN       EQU     32        Length of Prli
PRLI#SP        EQU     230       Subpool for the Prli
PRLI_LEN       EQU     *-PRLI
** BPXYPRLI End

```

## BPXYPTAT – Map attributes for pthread\_exit\_and\_get

VARLEN defines the number of bytes set aside to define the pthread attributes.

```

BPXYPTAT    VARLEN=1024
** BPXYPTAT: Pthread Attributes
** Used By:
PTAT        DSECT ,
PTATEYE     DC      C'BPXYPTAT'      Eye Catchter
PTATLENGTH  DC      A(PAT#LENGTH)    Length of PTAT
PTATSYSOFFSET DC    A(PATSYSOFFVAL)  Offset of SYSATTRS
PTATSYSENGTH DC    A(PATSYSENGVAL)   Length of SYSATTRS
PTATUSEROFFSET DC    A(PATUSEROFFVAL) Offset of USERATTRS
PTATUSERLENGTH DC    A(L'PTATUSERATTRS) Length of USERATTRS
PTATSYSOFFVAL EQU    *-PTAT Offset value of System Attribute Area
PTATSYSATTRS DS      0F System attributes
PTATDETACHSTATE DS    F Detach State of thread to be created:
PTATUNDETACHED EQU    0
PTATDETACHED EQU    1
PTATWEIGHT  DS      F Weight of thread to be created:
PTATHEAVY   EQU    0
PTATMEDIUM EQU    1
PTATSYNCTYPE DS      F Synchronous processing type of thread:
PTATSYNCHRONOUS EQU    0
PTATASYNCHRONOUS EQU    1 /*
PTATSHSPMASK DS      0XL16 /*
DS          XL15 /*
PTATSHSPBYTE16 DS      XL1 /*
PTATSHSPINUSE EQU    X'01' 0=system default used
*                1=use mask
*                default shared subpools 1, 2, 78
PTATSYSENGVAL EQU    *-PTATSYSENGATTRS Length of System Attributes
PTATUSEROFFVAL EQU    *-PTAT Offset of user attribute area
PTATUSERATTRS DS      CL1024 User attributes area
PTAT#LENGTH EQU    *-PTAT Length of this structure
** BPXYPTAT End

```

## BPXYPTRC – Map parameters for ptrace

VARLEN defines the number of bytes needed to hold the path name. The default is the maximum path name, 1024.

```

BPXYPTRC
*
* *****
* *
* * Ptrace PT_LDINFO return structure. Note that this maps one *
* * element, corresponding to one load module. Each element *
* * consists of a fixed portion, and a variable portion (the path *
* * name and member name character strings). The character strings *
* * are terminated with a null value (X'00'). Each loader info *
* * element immediately follows the last null terminator for the *
* * previous element. The first full word of each element is an *
* * offset to the next element. Thus, the start of the next element *
* * can be specified as follows: *
* *
* *      NextLDInfo = Addr(PtLDInfo)+PtLDInfoNext *
* *
* *****
*
*
PtLDInfo      DSECT
PtLDFixedArea DS      CL32
ORG          PtLDFixedArea
PtLDInfoNext  DS      F Offset to next element
PtLDInfoFD    DS      F File descriptor for this load module (not used) X
PtLDTextOrg   DS      A Program text origin address (i.e. X load point address)
PtLDTextSize  DS      F Length of text
PtLDTextSubpool DS    CL1 Subpool where text is loaded
PtLDTextFlags DS      B Text related flags
PtLDTextWrite EQU    X'80' 0 = text can be read but not X
                        written into 1 = text can be read X
                        and written into

```

```

PtLDTextMVS      EQU  X'40'  0 = File system load module 1 =   X
                                MVS load module
PtLDTextExt      EQU  X'20'  0 = Only 1 text extent 1 = More   X
                                than one text extent. First extent X
                                is in this element, extent 2 - n   X
                                are in the PtLDInfoExt area
PtLD0ffExt       DS      H      Offset from this element to      X
                                element extension. 0 if there is   X
                                no extension for this element
PtLDData0rg      DS      A      Program data origin address (not   X
                                used)
PtLDDataSize     DS      F      Length of data (not used)
PtLDDataSubpool  DS      X      Subpool where data is loaded (not   X
                                used)
PtLDDataFlags    DS      B      Data related flags (not used)
                                DS      H      Reserved
PtLDVarArea      DS      0C
PtLDPathName     DS      0C      Fully qualified path name of load   X
                                module
PtLDMemberName   DS      0C      Member name of load module (not    X
                                used)
PtLDInfo_Len     EQU      *-PtLDInfo
*
*      PT_LD64Info area, to succeed the one above
*
*
PtLD64Info       DSECT
PtLD64FixedArea  DS      CL32
ORG      PtLD64FixedArea
PtLD64InfoNext   DS      F      Offset to next element
                                DS      F      Reserved
PtLD64TextOrg    DS      AD      Program text origin address (i.e.   X
                                load point address)
PtLD64TextSize   DS      FL8      Length of text
PtLD64TextSubpool DS      CL1      Subpool where text is loaded
PtLD64TextFlags  DS      B      Text related flags
PtLD64TextWrite  EQU      X'80'  0 = text can be read but not      X
                                written into 1 = text can be read   X
                                and written into
PtLD64TextMVS    EQU      X'40'  0 = File system load module 1 =   X
                                MVS load module
PtLD64TextExt    EQU      X'20'  0 = Only 1 text extent 1 = More   X
                                than one text extent. First extent X
                                is in this element, extent 2 - n   X
                                are in the PtLDInfoExt area
PtLD640ffExt     DS      H      Offset from this element to      X
                                element extension. 0 if there is   X
                                no extension for this element
                                DS      F      Reserved
PtLD64VarArea    DS      0C
PtLD64PathName   DS      0C      Fully qualified path name of load   X
                                module
PtLD64Info_Len   EQU      *-PtLD64Info
PtLDInfoExt      DSECT
PtLDInfoNumTextExt DS      H      Number of additional text extents X
                                in the following arrays that are   X
                                meaningful, up to 15 in this area, X
                                for a total of 16
                                reserved
PtLDTextOrgExt   DS      H      reserved
                                DS      A      Program text origin address (i.e.   X
                                load point address)
PtLDTextSizeExt  ORG      PtLDTextOrgExt+60
                                DS      F      Length of text
                                ORG      PtLDInfoExt+124
PtLDInfoExt_Len  EQU      *-PtLDInfoExt
PtLD64InfoExt    DSECT
PtLD64InfoNumTextExt DS      F      Number of additional text extents X
                                in the following arrays that are   X
                                meaningful, up to 15 in this area, X
                                for a total of 16
                                reserved
PtLD64TextOrgExt DS      F      reserved
                                DS      AD      Program text origin address (i.e.   X
                                load point address)
PtLD64TextSizeExt ORG      PtLD64TextOrgExt+120
                                DS      FL8      Length of text
PtLDNullTerm     EQU      0      Null terminator for character      X
                                strings
PtLD64InfoExt_Len ORG      PtLD64InfoExt+248
                                EQU      *-PtLD64InfoExt
*
* *****
* *

```

```

* * Ptrace thread information return structure. Note that this *
* * maps one element, corresponding to one thread. *
* * *
* * Note: the only valid information for a dead thread is: *
* * *
* * PTPTNEXT, PTPTHID, PTPSTATEACTIVE=0, PTPTKERNELPTHREAD, *
* * PTPTEXTSTATUS *
* *****
*
*
PtPTINFO          DSECT
PtptNext          DS      A      Offset to next element
PtptTHID          DS      CL8     Thread ID
PtptReserved      DS      CL16    Reserved
PtptState         DS      BL4     Thread state flags
PtptState1        ORG      PtptState
PtptStateActive   EQU      X'80'
*
* *****
* *
* * 0 = thread is dead
* * 1 = thread is active
* *
* *****
*
*
PtptStateAsync    EQU      X'40'
*
* *****
* *
* * 1 = thread is asynchronous
* * (is also active but not
* * yet running)
* *
* *****
*
*
PtptStateCancelPend EQU      X'20'
*
* *****
* *
* * 1 = cancel is pending
* *
* *****
*
*
PtptState2        DS      B      Thread state flag byte
PtptState3        DS      B      Thread state flag byte
PtptState4        DS      B      Thread state flag byte
PtptKernelAttr    DS      BL4     Thread kernel attributes
PtptKernel1       ORG      PtptKernelAttr
PtptKernelDetach  DS      B      Thread kernel attribute byte
PtptKernelDetach  EQU      X'80'
*
* *****
* *
* * 0 = thread is not detached
* * 1 = thread is detached
* *
* *****
*
*
PtptKernelMedium  EQU      X'40'
*
* *****
* *
* * 0 = thread is heavyweight
* * 1 = thread is mediumweight
* *
* *****
*
*
PtptKernelAsync   EQU      X'20'
*
* *****
* *
* * 0 = thread is synchronous
* * 1 = thread is asynchronous
* *
* *****
*

```

```

*
PtpKernelPthread      EQU   X'10'
*
* *****
* *
* * 1 = thread is created via
* * pthread create
* *
* *****
*
*
PtpKernel2            DS    B      Thread kernel attribute byte
PtpKernelHold         EQU   X'80'
*
* *****
* *
* * 1 = thread is held
* *
* *****
*
*
PtpKernel3            DS    B      Reserved
PtpKernel4            DS    B      Reserved
PtpExitStatus         DS    CL4
*
* *****
* *
* * Thread exit status if dead
* * for amode 31 pthread or
* * Low part of thread exit
* * status
* * for amode 64 pthread
* * (PtpStateActive = 0)
* *
* *****
*
*
PtpPendingSigmask     DS     BL8
*
* *****
* *
* * Mask of pending signals
* * (bit 0 represents signal 1)
* * (bit 63 represents signal 64)
* *
* *****
*
*
PtpExitStatusHigh     DS     CL4
*
* *****
* *
* * Thread exit status if dead
* * High part of thread exit
* * status
* * for amode 64 pthread
* * (PtpStateActive = 0)
* *
* *****
*
*
* DS     CL4      reserved
PtpTINFO_Len          EQU   *-PtpTINFO
*
* *****
* *
* * Ptrace thread information extended structure. Note that this
* * maps one element, corresponding to one thread. PtpxInfo maps
* * exactly to PtpTInfo
* *
* *
* * Note: the only valid information for a dead thread is:
* *
* *   PTPXNEXT, PTPXTHID, PTPXSTATEACTIVE=0, PTPXKERNELPTHREAD,
* *   PTPXEXITSTATUS
* *
* *****
*
*
PtpPhINFO             DSECT      PT_THREAD_INFO_EXTENDED header
                        information
PtpPhInfoBase         DS     CL56

```



```
*
*****
*
* PtpInfoBase contains information about the process and
* pointers to the next array of thread info
*
*****
*
ORG    PtPhInfoBase
PtPhId DS CL4 Acronym
PTPhNext DS A Address of the next PtPhInfo
PtPhPtxOff DS A Offset of first Ptpx in this chunk X
of storage
PTPhPid DS F Process id of the threads
PTPhPendingSigmask DS BL8 Signals pending at the process
PTPhBlockedSigmask DS BL8 blocked signals at process
PtPhThreadNum DS F Total number of threads reported X
in chain
PtPhTnum DS CL4 Threads in the Current buffer
PtPhPtxLen DS F Length of the Ptpx in this buffer
DS CL4 Reserved
ORG PtPhINFO+56
PtPhINFO_Len EQU *-PtPhINFO
PtPxINFO DSECT PT_THREAD_INFO_EXTENDED maps a X
single entry
PtPxInfoBase DS CL64
ORG PtPxInfoBase
PTPXNext DS A Offset to next element
PTPXTHID DS CL8 Thread ID
PTPXTcb DS A Tcb address for this process
PTPXOtcB DS A OtcB address for this process
PTPXBlockedSigmask DS BL8 blocked signals
PTPXState DS BL4 Thread state flags
ORG PTPXState
PTPXState1 DS B Thread state flag byte
PTPXStateActive EQU X'80' 0 = thread is dead
*
*****
*
* 1 = thread is active
*
*****
*
PTPXStateAsync EQU X'40' 1 = N/A
*
*****
*
* (is also active but not
* yet running)
*
*****
*
PTPXStateCancelPend EQU X'20'
*
*****
*
* 1 = cancel is pending
*
*****
*
PTPXState2 DS B Thread state flag byte
PTPXState3 DS B Thread state flag byte
PTPXState4 DS B Thread state flag byte
PTPXKernelAttr DS BL4 Thread kernel attrites
ORG PTPXKernelAttr
PTPXKernel1 DS B Thread kernel attribute byte
PTPXKernelDetach EQU X'80'
*
*****
*
* 0 = thread is not detached
* 1 = thread is detached
*
*****
*
PTPXKernelMedium EQU X'40'
```

```

* *****
* *
* * 0 = thread is heavyweight
* * 1 = thread is mediumweight
* *
* *****
*
*
PTPXKernelAsync      EQU   X'20'
*
* *****
* *
* * 0 = thread is synchronous
* * 1 = thread is asynchronous
* *
* *****
*
*
PTPXKernelPthread    EQU   X'10'
*
* *****
* *
* * 1 = thread is created via
* * pthread create
* *
* *****
*
*
PTPXKernel2          DS     B      Thread kernel attribute byte
PTPXKernelHold        EQU   X'80'
*
* *****
* *
* * 1 = thread is held
* *
* *****
*
*
PTPXKernel3          DS     B      Reserved
PTPXKernel4          DS     B      Reserved
PTPXExitStatus        DS     CL4
*
* *****
* *
* * Thread exit status if dead
* * for amode 31 pthread or
* * Low part of thread exit
* * status
* * for amode 64 pthread
* * (PTPXStateActive = 0)
* *
* *****
*
*
PTXPendingSigmask    DS     BL8    Mask of pending signals is set) X
                                (bit 0 represents signal 1)
PTXPid               DS     F      Process id for this Thid
PTXAsid              DS     H
PTPXFlags            DS     BL2    Thread related flags
PtPxIpt              EQU   X'80'   Ipt Thread
PtPxInComplete        EQU   X'40'   The reported thread information is X
                                incomplete
PTPX0apb             DS     A      Pointer to the 0apb
PTPXExitStatusHigh    DS     CL4
PtPxINFO_Len         EQU   *-PtPxINFO
*
* *****
* *
* * For a dead thread
* * (PTPXStateActive = 0)
* * if AMODE 64 pthread this is
* * the high half of the thread
* * exit status. If an AMODE 31
* * pthread this field is unused
* *
* *****
*
* *****
* *
* * Ptrace explain information return structure.
* *
* *****

```

```

*
*
PtExInfo          DSECT
PtExInfo31        DS    CL12
                  ORG    PtExInfo31
PtExReg1          DS    F      Register 1 at CEEVDBG entry
PtExReg12         DS    F      Register 12 at CEEVDBG entry
PtExReg13         DS    F      Register 13 at CEEVDBG entry
                  DS    F      reserved
PtExInfo64        DS    CL24
                  ORG    PtExInfo64
PtExG64R1         DS    FL8     Register 1 at CEEVDBG entry
PtExG64R12        DS    FL8     Register 12 at CEEVDBG entry
PtExG64R13        DS    FL8     Register 13 at CEEVDBG entry
PtExInfo_Len      EQU    *-PtExInfo
*
* *****
* *
* * Ptrace program recovery parameters structure.
* * (This area is provided by the caller)
* *
* *****
*
*
PtPICParms        DSECT
PtPICRegisters    DS    A      Address of GPRs at time of
                              interrupt
PtPICPSW          DS    A      Address of PSW at time of interrupt
PtPICIntCode      DS    H      Program interrupt code
PtPICSigNumber    DS    H      Return value indicating signal
                              number that should be raised by the
                              caller if the PtPICUseSigNum flag
                              is set
PtPICFlags        DS    BL4     Flags
PtPICICModified   EQU    X'80'  0 = The instruction counter portion
                              of the PSW pointed to by the
                              PtPICPSW field has not been
                              modified 1 = The instruction
                              counter portion of the PSW pointed
                              to by the PtPICPSW field has been
                              modified - continue execution at
                              this modified address
PtPICRegsModified EQU    X'40'  0 = The registers pointed to by the
                              PtPICRegisters field have not been
                              modified 1 = The registers pointed
                              to by the PtPICRegisters field have
                              been modified
PtPICUseSigNum    EQU    X'20'  1 = Raise the signal number
                              returned in the PtPICSigNumber
                              field upon return
PtPICBypassSig    EQU    X'10'  1 = Do not raise any signal upon
                              return
PtPICILCExists    EQU    X'08'  1 = PtPICILC field is present
PtPICHiRegsExists EQU    X'04'  1 = The PtPICHiRegisters field is
                              present
PtPICHiRegsModified EQU    X'02' 0 = The registers pointed to by the
                              PtPICHiRegisters field have not
                              been modified 1 = The registers
                              pointed to by the PtPICHiRegisters
                              field have been modified
PtPICAmode64      EQU    X'01'  Use 64 bit addresses for PSW and
                              registers
                  DS    BL3     Reserved
                  ORG    PtPICFlags+4
PtPICAbendCode    DS    BL4     Abend code or zero
                  ORG    PtPICAbendCode
PtPICAbendFlags   DS    B      Abend code flags
PtPICAbendCC      DS    BL3     System completion code (first 12
                              bits) and user completion code
                              (second 12 bits)
PtPICAbendReason  DS    F      Abend reason code or zero
PtPICILC          DS    X      Instruction length code (only
                              present if PtPICILCExists flag is
                              set)
PtPICReserved     DS    CL3     Reserved
PtPICHiRegisters  DS    A      Address of high GPRs at time of
                              interrupt
PtPICRegisters64  DS    AD      Address of GPRs at time of
                              interrupt - Amode 64
PtPICPSW64        DS    AD      Address of PSW at time of interrupt
                              - Amode 64
PtPICHiRegisters64 DS    AD      Address of high GPRs at time of

```

PtPICRsvd	DS	CL8	interrupt - Amode 64 Reserved. This area is provided by the caller and may not be present in old releases of code
* * ***** * * * * Ptrace request parameter definitions. * * * ***** * *			
PT_TRACE_ME	EQU	0	Debug this process
PT_READ_I	EQU	1	Read a full word
PT_READ_D	EQU	2	Read a full word
PT_READ_U	EQU	3	Read control info
PT_WRITE_I	EQU	4	Write a full word
PT_WRITE_D	EQU	5	Write a full word
PT_CONTINUE	EQU	7	Continue the process
PT_KILL	EQU	8	Terminate the process
PT_READ_GPR	EQU	11	Read GPR, CR, PSW
PT_READ_FPR	EQU	12	Read FPR
PT_READ_VR	EQU	13	Read VR
PT_WRITE_GPR	EQU	14	Write GPR, CR, PSW
PT_WRITE_FPR	EQU	15	Write FPR
PT_WRITE_VR	EQU	16	Write VR
PT_READ_BLOCK	EQU	17	Read storage
PT_WRITE_BLOCK	EQU	19	Write storage
PT_READ_GPRH	EQU	20	Read GPRH
PT_WRITE_GPRH	EQU	21	Write GPRH
PT_REGHSET	EQU	22	Read all GPRHs
PT_ATTACH	EQU	30	Attach to a process
PT_DETACH	EQU	31	Detach from a process
PT_REGSET	EQU	32	Read all GPRs
PT_REATTACH	EQU	33	Reattach to a process
PT_LDINFO	EQU	34	Read loader info
PT_MULT	EQU	35	Multi process mode
PT_LD64INFO	EQU	36	RMODE64 Info Area
PT_BLOCKREQ	EQU	40	Block request
PT_THREAD_INFO	EQU	60	Read thread info
PT_THREAD_MODIFY	EQU	61	***** ***** Modify thread kernel information ***** *****
PT_THREAD_READ_FOCUS	EQU	62	***** ***** Read current focus thread ID ***** *****
PT_THREAD_WRITE_FOCUS	EQU	63	***** ***** Modify current focus thread ID ***** *****
PT_THREAD_HOLD	EQU	64	***** ***** Modify thread hold state ***** *****
PT_THREAD_SIGNAL	EQU	65	***** ***** Queue a signal for a thread ***** *****
PT_EXPLAIN	EQU	66	***** ***** Return extended event info ***** *****
PT_EVENTS	EQU	67	***** ***** Modify extended events list ***** *****
PT_THREAD_INFO_EXTENDED	EQU	68	***** ***** Read extended thread info ***** *****
PT_REATTACH2	EQU	71	***** ***** Reattach

```

to a process (extended) *****
*****
*****
PT_CAPTURE          EQU    72      *****
***** Capture
debugged storage *****
*****
PT_UNCAPTURE        EQU    73      *****
***** Uncapture
debugged storage *****
*****
PT_GET_THREAD_TCB    EQU    74      *****
***** Get TCB
address of a thread *****
*****
PT_GET_ALET          EQU    75      *****
***** Get Alet
of target PID *****
*****
PT_SWAPIN            EQU    76      *****
***** SWAPIN
target PID's A.S. *****
*****
PT_EXTENDED_EVENT    EQU    98      *****
***** Debug an
extended event *****
*****
PT_RECOVER           EQU    99      Debug a program check
*
* *****
* *
* * Ptrace register definitions. The following are defined:
* *
* * - General purpose registers
* * - Floating point registers
* * - PSW registers
* * - Control registers
* * - General Purpose High Registers
* *
* *****
*
*
PT_GPR0              EQU    0      General purpose register 0
PT_GPR1              EQU    1      General purpose register 1
PT_GPR2              EQU    2      General purpose register 2
PT_GPR3              EQU    3      General purpose register 3
PT_GPR4              EQU    4      General purpose register 4
PT_GPR5              EQU    5      General purpose register 5
PT_GPR6              EQU    6      General purpose register 6
PT_GPR7              EQU    7      General purpose register 7
PT_GPR8              EQU    8      General purpose register 8
PT_GPR9              EQU    9      General purpose register 9
PT_GPR10             EQU    10     General purpose register 10
PT_GPR11             EQU    11     General purpose register 11
PT_GPR12             EQU    12     General purpose register 12
PT_GPR13             EQU    13     General purpose register 13
PT_GPR14             EQU    14     General purpose register 14
PT_GPR15             EQU    15     General purpose register 15
PT_FPR0              EQU    16     Floating point register 0
PT_FPR1              EQU    17     Floating point register 1
PT_FPR2              EQU    18     Floating point register 2
PT_FPR3              EQU    19     Floating point register 3
PT_FPR4              EQU    20     Floating point register 4
PT_FPR5              EQU    21     Floating point register 5
PT_FPR6              EQU    22     Floating point register 6
PT_FPR7              EQU    23     Floating point register 7
PT_FPR8              EQU    24     Floating point register 8
PT_FPR9              EQU    25     Floating point register 9
PT_FPR10             EQU    26     Floating point register 10
PT_FPR11             EQU    27     Floating point register 11
PT_FPR12             EQU    28     Floating point register 12
PT_FPR13             EQU    29     Floating point register 13
PT_FPR14             EQU    30     Floating point register 14
PT_FPR15             EQU    31     Floating point register 15
PT_FPC               EQU    32     Floating point control register
PT_PSW               EQU    40     PSW

```

```

PT_PSW0      EQU 40      Left half of the PSW
PT_PSW1      EQU 41      Right half of the PSW
PT_CR0       EQU 42      Control register 0
PT_CR1       EQU 43      Control register 1
PT_CR2       EQU 44      Control register 2
PT_CR3       EQU 45      Control register 3
PT_CR4       EQU 46      Control register 4
PT_CR5       EQU 47      Control register 5
PT_CR6       EQU 48      Control register 6
PT_CR7       EQU 49      Control register 7
PT_CR8       EQU 50      Control register 8
PT_CR9       EQU 51      Control register 9
PT_CR10      EQU 52      Control register 10
PT_CR11      EQU 53      Control register 11
PT_CR12      EQU 54      Control register 12
PT_CR13      EQU 55      Control register 13
PT_CR14      EQU 56      Control register 14
PT_CR15      EQU 57      Control register 15
PT_GPRH0     EQU 58      GP High register 0
PT_GPRH1     EQU 59      GP High register 1
PT_GPRH2     EQU 60      GP High register 2
PT_GPRH3     EQU 61      GP High register 3
PT_GPRH4     EQU 62      GP High register 4
PT_GPRH5     EQU 63      GP High register 5
PT_GPRH6     EQU 64      GP High register 6
PT_GPRH7     EQU 65      GP High register 7
PT_GPRH8     EQU 66      GP High register 8
PT_GPRH9     EQU 67      GP High register 9
PT_GPRH10    EQU 68      GP High register 10
PT_GPRH11    EQU 69      GP High register 11
PT_GPRH12    EQU 70      GP High register 12
PT_GPRH13    EQU 71      GP High register 13
PT_GPRH14    EQU 72      GP High register 14
PT_GPRH15    EQU 73      GP High register 15
PT_VR0       EQU 74      Vector register 0
PT_VR1       EQU 75      Vector register 1
PT_VR2       EQU 76      Vector register 2
PT_VR3       EQU 77      Vector register 3
PT_VR4       EQU 78      Vector register 4
PT_VR5       EQU 79      Vector register 5
PT_VR6       EQU 80      Vector register 6
PT_VR7       EQU 81      Vector register 7
PT_VR8       EQU 82      Vector register 8
PT_VR9       EQU 83      Vector register 9
PT_VR10      EQU 84      Vector register 10
PT_VR11      EQU 85      Vector register 11
PT_VR12      EQU 86      Vector register 12
PT_VR13      EQU 87      Vector register 13
PT_VR14      EQU 88      Vector register 14
PT_VR15      EQU 89      Vector register 15
PT_VR16      EQU 90      Vector register 16
PT_VR17      EQU 91      Vector register 17
PT_VR18      EQU 92      Vector register 18
PT_VR19      EQU 93      Vector register 19
PT_VR20      EQU 94      Vector register 20
PT_VR21      EQU 95      Vector register 21
PT_VR22      EQU 96      Vector register 22
PT_VR23      EQU 97      Vector register 23
PT_VR24      EQU 98      Vector register 24
PT_VR25      EQU 99      Vector register 25
PT_VR26      EQU 100     Vector register 26
PT_VR27      EQU 101     Vector register 27
PT_VR28      EQU 102     Vector register 28
PT_VR29      EQU 103     Vector register 29
PT_VR30      EQU 104     Vector register 30
PT_VR31      EQU 105     Vector register 31
PT_PSWG      EQU 106     PSWG
PT_PSWG0     EQU 106     Bytes 0-3
PT_PSWG1     EQU 107     Bytes 4-7
PT_PSWG2     EQU 108     Bytes 8-11 (IA high word)
PT_PSWG3     EQU 109     Bytes 12-15 (IA low word)
*
* *****
* *
* * Ptrace User Area offset definitions. Offsets for signal catcher *
* * information are defined by the limits below. Any offset between *
* * the minimum and maximum signal numbers is a request for signal *
* * catcher information for that signal number (i.e. offset 3 means *
* * signal catcher information for signal number 3). *
* * *
* *****
*

```

```

*
PtUArea#MinSig      EQU 1      Lowest signal number
PtUArea#MaxSig      EQU 1024   Highest signal number
PtUArea#IntCode     EQU 1025   Request for program interrupt code
PtUArea#AbendCC     EQU 1026   Request for abend completion code
PtUArea#AbendRC     EQU 1027   Request for abend reason code
PtUArea#SigCode     EQU 1028   Request for signal code
PtUArea#ILC         EQU 1029   Request for instruction length code
PtUArea#PrFlags     EQU 1030   Request for process flags
*
* *****
* * Ptrace miscellaneous definitions.
* *
* *****
*
*
PtContNorm          EQU 1      Continue normally (continue address
                                not changed) for a PT_CONTINUE
                                request
PtNoStickyPgm       EQU 1      Main program of process is not
                                sticky bit program. Returned on
                                PTUAREA#PRFLAGS request
PtMaximumLength     EQU 64000  Maximum storage length
PtLD#FixedLen       EQU 32     Length of PtLDInfo fixed area
PtLD64#FixedLen     EQU 32     Length of PtLD64Info fixed area
PtPT#Length         EQU 56     Length of PtpTInfo fixed area
PtPh#Length         EQU 56     Length of PtPhInfo fixed area on
                                double word boundary
PtPx#Length         EQU 64     Length of PtpxInfo fixed area on
                                double word boundary
PtEx#Length         EQU 40     Length of PtExInfo
PtEx31#Length       EQU 12     Length of PtExInfo 31 bit
PtPIC#Length1       EQU 28     Length of PtPicParms if
                                PtPicHiRegsExists = OFF
PtPIC#Length2       EQU 32     Length of PtPicParms if
                                PtPicHiRegsExists = ON
PtPIC#Length        EQU 64     Length of PtPICParms
PtPICParms_Len      EQU *-PtPICParms
*
* *****
* * Ptrace PT_BlockReq structure. This request allows the user to
* * block several different Ptrace requests into a single call to
* * Ptrace. The block request structures mapped below must be
* * contained in a single large area. This area is pointed to by
* * the Ptrace Address parameter and its length is contained in the
* * Ptrace Data parameter. The PtBRInfo structure must be at offset
* * zero into the provided area.
* *
* * Offsets are used to locate all relevant areas so that the Ptrace
* * block request input may be relocated. All offsets are relative
* * to the main input, the PtBRInfo area. A given request block,
* * such as the PtBR_GPR structure, may be found using the
* * PtBRInfo address + PtBRReqBlkOff(x).
* *
* * Only certain requests may be blocked into a single call to
* * Ptrace. The requests that may be blocked are --
* * PT_READ_GPR
* * PT_WRITE_GPR
* * PT_READ_FPR
* * PT_WRITE_FPR
* * PT_READ_GPRH
* * PT_WRITE_GPRH
* * PT_READ_U
* * PT_READ_D
* * PT_READ_I
* * PT_WRITE_D
* * PT_WRITE_I
* * PT_READ_BLOCK
* * PT_WRITE_BLOCK.
* *
* *****
*
*
PtBRInfo            DSECT
PtBRFixedArea       DS CL16
ORG PtBRFixedArea
PtBRNumReqs         DS F      Number of requests in PtBRReqs
DS CL12             Reserved
PtBRReqs            DS CL16   requests
ORG PtBRReqs

```

PtBRType	DS	F	Type of request. For example, PT_READ_BLOCK. This entry is ignored if this field is zero
PtBRStatus	DS	F	Status from request. Same as reasoncode on individual call of same type
PtBRReqBlkOff	DS	F	Offset to request block further defining request and whose format is dependant on the request type
PtBRInfo_Len	DS	CL4	reserved
	EQU		*-PtBRInfo
* * ***** * * Structure for PT_Read_GPR and PT_Write_GPR. * * ***** * *			
PtBR_GPR	DSECT		
PtBR_GPR_CntlGPR	DS	BL2	Only used on write request
	ORG		PtBR_GPR_CntlGPR
PtBR_GPR_CntlGPR1	DS	B	
PtBR_GPR_WGPR0	EQU	X'80'	Write content of GPR 0
PtBR_GPR_WGPR1	EQU	X'40'	Write content of GPR 1
PtBR_GPR_WGPR2	EQU	X'20'	Write content of GPR 2
PtBR_GPR_WGPR3	EQU	X'10'	Write content of GPR 3
PtBR_GPR_WGPR4	EQU	X'08'	Write content of GPR 4
PtBR_GPR_WGPR5	EQU	X'04'	Write content of GPR 5
PtBR_GPR_WGPR6	EQU	X'02'	Write content of GPR 6
PtBR_GPR_WGPR7	EQU	X'01'	Write content of GPR 7
PtBR_GPR_CntlGPR2	DS	B	
PtBR_GPR_WGPR8	EQU	X'80'	Write content of GPR 8
PtBR_GPR_WGPR9	EQU	X'40'	Write content of GPR 9
PtBR_GPR_WGPR10	EQU	X'20'	Write content of GPR 10
PtBR_GPR_WGPR11	EQU	X'10'	Write content of GPR 11
PtBR_GPR_WGPR12	EQU	X'08'	Write content of GPR 12
PtBR_GPR_WGPR13	EQU	X'04'	Write content of GPR 13
PtBR_GPR_WGPR14	EQU	X'02'	Write content of GPR 14
PtBR_GPR_WGPR15	EQU	X'01'	Write content of GPR 15
	ORG		PtBR_GPR_CntlGPR+2
PtBR_GPR_CntlMisc	DS	BL2	Only used on write request
PtBR_GPR_WPSW	EQU	X'80'	Write content of PSW, word 2
PtBR_GPR_WPSWG	EQU	X'40'	Write content of PSWG
PtBR_GPR_PSWG_Req	EQU	X'20'	request large psw
	DS	CL12	Reserved
PtBR_GPR_GPRs	DS	CL64	General purpose registers
	ORG		PtBR_GPR_GPRs
PtBR_GPR_GPR00	DS	F	GPR 00
PtBR_GPR_GPR01	DS	F	GPR 01
PtBR_GPR_GPR02	DS	F	GPR 02
PtBR_GPR_GPR03	DS	F	GPR 03
PtBR_GPR_GPR04	DS	F	GPR 04
PtBR_GPR_GPR05	DS	F	GPR 05
PtBR_GPR_GPR06	DS	F	GPR 06
PtBR_GPR_GPR07	DS	F	GPR 07
PtBR_GPR_GPR08	DS	F	GPR 08
PtBR_GPR_GPR09	DS	F	GPR 09
PtBR_GPR_GPR10	DS	F	GPR 10
PtBR_GPR_GPR11	DS	F	GPR 11
PtBR_GPR_GPR12	DS	F	GPR 12
PtBR_GPR_GPR13	DS	F	GPR 13
PtBR_GPR_GPR14	DS	F	GPR 14
PtBR_GPR_GPR15	DS	F	GPR 15
PtBR_GPR_CRs	DS	CL64	Control registers. May be read but will not be written
	ORG		PtBR_GPR_CRs
PtBR_GPR_CR00	DS	F	CR 00
PtBR_GPR_CR01	DS	F	CR 01
PtBR_GPR_CR02	DS	F	CR 02
PtBR_GPR_CR03	DS	F	CR 03
PtBR_GPR_CR04	DS	F	CR 04
PtBR_GPR_CR05	DS	F	CR 05
PtBR_GPR_CR06	DS	F	CR 06
PtBR_GPR_CR07	DS	F	CR 07
PtBR_GPR_CR08	DS	F	CR 08
PtBR_GPR_CR09	DS	F	CR 09
PtBR_GPR_CR10	DS	F	CR 10
PtBR_GPR_CR11	DS	F	CR 11
PtBR_GPR_CR12	DS	F	CR 12
PtBR_GPR_CR13	DS	F	CR 13
PtBR_GPR_CR14	DS	F	CR 14
PtBR_GPR_CR15	DS	F	CR 15
PtBR_GPR_PSW	DS	CL8	PSW. May be read but only the



```

                                rightmost 4 bytes (word 2) will be
                                written
PtBR_GPR_PSW_W1      ORG   PtBR_GPR_PSW
PtBR_GPR_PSW_W2      DS    CL4    PSW word 1.
PtBR_GPR_PSW_W2      DS    CL4    PSW word 2
PtBR_GPR_Len         EQU    *-PtBR_GPR
PtBR_GPR_EXT         DSECT
DS                   CL152  The first part of the union, the
                                old struct

PtBR_GPR_PSWG        DS    CL16
PtBR_GPR_PSWG_W1     ORG    PtBR_GPR_PSWG
PtBR_GPR_PSWG_W1     DS    CL4
PtBR_GPR_PSWG_W2     DS    CL4
PtBR_GPR_PSWG_W3     DS    CL4
PtBR_GPR_PSWG_W4     DS    CL4
PtBR_GPR_EXT_Len     EQU    *-PtBR_GPR_EXT
*
* *****
* * Structure for PT_Read_FPR and PT_Write_FPR.
* *****
*
*
PtBR_FPR             DSECT
PtBR_FPR_CntlFPR     DS    BL2    Only used on write request
PtBR_FPR_CntlFPR     ORG    PtBR_FPR_CntlFPR
PtBR_FPR_CntlFPR1    DS    B
PtBR_FPR_WFPR0       EQU    X'80'  Write content of FPR 0
PtBR_FPR_WFPR1       EQU    X'40'  Write content of FPR 1
PtBR_FPR_WFPR2       EQU    X'20'  Write content of FPR 2
PtBR_FPR_WFPR3       EQU    X'10'  Write content of FPR 3
PtBR_FPR_WFPR4       EQU    X'08'  Write content of FPR 4
PtBR_FPR_WFPR5       EQU    X'04'  Write content of FPR 5
PtBR_FPR_WFPR6       EQU    X'02'  Write content of FPR 6
PtBR_FPR_WFPR7       EQU    X'01'  Write content of FPR 7
PtBR_FPR_CntlFPR2    DS    B
PtBR_FPR_WFPR8       EQU    X'80'  Write content of FPR 8
PtBR_FPR_WFPR9       EQU    X'40'  Write content of FPR 9
PtBR_FPR_WFPR10      EQU    X'20'  Write content of FPR 10
PtBR_FPR_WFPR11      EQU    X'10'  Write content of FPR 11
PtBR_FPR_WFPR12      EQU    X'08'  Write content of FPR 12
PtBR_FPR_WFPR13      EQU    X'04'  Write content of FPR 13
PtBR_FPR_WFPR14      EQU    X'02'  Write content of FPR 14
PtBR_FPR_WFPR15      EQU    X'01'  Write content of FPR 15
PtBR_FPR_CntlFPR2    ORG    PtBR_FPR_CntlFPR+2
PtBR_FPR_CntlMisc    DS    BL2    Only used on write request
PtBR_FPR_WFPC        EQU    X'80'  Write content of FPC
PtBR_FPR_WFPC        DS    CL12    Reserved
PtBR_FPR_FPRs        DS    CL128  Floating point registers
PtBR_FPR_FPRs        ORG    PtBR_FPR_FPRs
PtBR_FPR_FPR00       DS    CL8    FPR 00
PtBR_FPR_FPR01       DS    CL8    FPR 01
PtBR_FPR_FPR02       DS    CL8    FPR 02
PtBR_FPR_FPR03       DS    CL8    FPR 03
PtBR_FPR_FPR04       DS    CL8    FPR 04
PtBR_FPR_FPR05       DS    CL8    FPR 05
PtBR_FPR_FPR06       DS    CL8    FPR 06
PtBR_FPR_FPR07       DS    CL8    FPR 07
PtBR_FPR_FPR08       DS    CL8    FPR 08
PtBR_FPR_FPR09       DS    CL8    FPR 09
PtBR_FPR_FPR10       DS    CL8    FPR 10
PtBR_FPR_FPR11       DS    CL8    FPR 11
PtBR_FPR_FPR12       DS    CL8    FPR 12
PtBR_FPR_FPR13       DS    CL8    FPR 13
PtBR_FPR_FPR14       DS    CL8    FPR 14
PtBR_FPR_FPR15       DS    CL8    FPR 15
PtBR_FPR_FPC         DS    CL4    Floating Point Control Register
PtBR_FPR_Len         EQU    *-PtBR_FPR
*
* *****
* * Structure for PT_Read_GPRH PT_Write_GPRH.
* *****
*
*
PtBR_GPRH            DSECT
PtBR_GPRH_CntlGPRH   DS    BL2    Only used on write request
PtBR_GPRH_CntlGPRH   ORG    PtBR_GPRH_CntlGPRH
PtBR_GPRH_CntlGPRH1  DS    B
PtBR_GPRH_WGPRH0     EQU    X'80'  Write content of GPRH 0
PtBR_GPRH_WGPRH1     EQU    X'40'  Write content of GPRH 1
PtBR_GPRH_WGPRH2     EQU    X'20'  Write content of GPRH 2
PtBR_GPRH_WGPRH3     EQU    X'10'  Write content of GPRH 3
PtBR_GPRH_WGPRH4     EQU    X'08'  Write content of GPRH 4

```

```

PtBR_GPRH_WGPRH5      EQU  X'04'  Write content of GPRH 5
PtBR_GPRH_WGPRH6      EQU  X'02'  Write content of GPRH 6
PtBR_GPRH_WGPRH7      EQU  X'01'  Write content of GPRH 7
PtBR_GPRH_CntlGPRH2    DS      B
PtBR_GPRH_WGPRH8      EQU  X'80'  Write content of GPRH 8
PtBR_GPRH_WGPRH9      EQU  X'40'  Write content of GPRH 9
PtBR_GPRH_WGPRH10     EQU  X'20'  Write content of GPRH10
PtBR_GPRH_WGPRH11     EQU  X'10'  Write content of GPRH11
PtBR_GPRH_WGPRH12     EQU  X'08'  Write content of GPRH12
PtBR_GPRH_WGPRH13     EQU  X'04'  Write content of GPRH13
PtBR_GPRH_WGPRH14     EQU  X'02'  Write content of GPRH14
PtBR_GPRH_WGPRH15     EQU  X'01'  Write content of GPRH15
PtBR_GPRH_CntlMisc     DS      BL2  Only used on write request
                        DS      CL12  Reserved
PtBR_GPRH_GPRHs        DS      CL64  GP High registers.
                        ORG      PtBR_GPRH_GPRHs
PtBR_GPRH_GPRH00       DS      F      GPRH 00
PtBR_GPRH_GPRH01       DS      F      GPRH 01
PtBR_GPRH_GPRH02       DS      F      GPRH 02
PtBR_GPRH_GPRH03       DS      F      GPRH 03
PtBR_GPRH_GPRH04       DS      F      GPRH 04
PtBR_GPRH_GPRH05       DS      F      GPRH 05
PtBR_GPRH_GPRH06       DS      F      GPRH 06
PtBR_GPRH_GPRH07       DS      F      GPRH 07
PtBR_GPRH_GPRH08       DS      F      GPRH 08
PtBR_GPRH_GPRH09       DS      F      GPRH 09
PtBR_GPRH_GPRH10       DS      F      GPRH 10
PtBR_GPRH_GPRH11       DS      F      GPRH 11
PtBR_GPRH_GPRH12       DS      F      GPRH 12
PtBR_GPRH_GPRH13       DS      F      GPRH 13
PtBR_GPRH_GPRH14       DS      F      GPRH 14
PtBR_GPRH_GPRH15       DS      F      GPRH 15
                        DS      CL8    Reserved
PtBR_GPRH_Len          EQU  *-PtBR_GPRH
*
* *****
* * Structure for PT_Read_Block and PT_Write_Block. *
* *****
*
*
PtBR_Block              DSECT
PtBR_Block_AAddr        DS      A      address of area to read
PtBR_Block_ALen         DS      F      length of area to read
                        DS      CL8    Reserved
PtBR_Block_Buf          DS      0C      area to read into or write from.
                                      Must be at least PtBR_Block_ALen
                                      bytes large
PtBR_Block_Len          EQU  *-PtBR_Block
PtBR_Block64            DSECT
PtBR_Block_AAddr64      DS      AD     address of area to read
PtBR_Block_ALen64       DS      F      length of area to read
                        DS      CL4    Reserved
PtBR_Block_Buf64        DS      0C      area to read into or write from.
                                      Must be at least PtBR_Block_ALen
                                      bytes large
PtBR_Block64_Len        EQU  *-PtBR_Block64
*
* *****
* * Structure for PT_Read_D and PT_Write_D. *
* *****
*
*
PtBR_D                  DSECT
PtBR_D_Wordptr          DS      A      Address of fullword of data
PtBR_D_Word             DS      F      fullword of data at specified
                                      address for a read request or the
                                      data to be written to the specified
                                      address for a write request
PtBR_D_Len              EQU  *-PtBR_D
PtBR_D64                DSECT
PtBR_D_Wordptr64        DS      AD     Address of fullword of data
PtBR_D_Word64           DS      F      fullword of data at specified
                                      address for a read request or the
                                      data to be written to the specified
                                      address for a write request
PtBR_D64_Len            EQU  *-PtBR_D64
*
* *****
* * Structure for PT_Read_I and PT_Write_I. *
* *****
*

```

```

*
PtBR_I          DSECT
PtBR_I_Wordptr  DS    A      Address of fullword of program data
PtBR_I_Word     DS    F      fullword of program data at
                             specified address for a read
                             request or the program data to be
                             written to the specified address
                             for a write request

PtBR_I_Len      EQU    *-PtBR_I
PtBR_I64        DSECT
PtBR_I_Wordptr64 DS    AD     Address of fullword of program data
PtBR_I_Word64   DS    F      fullword of program data at
                             specified address for a read
                             request or the program data to be
                             written to the specified address
                             for a write request

PtBR_I64_Len    EQU    *-PtBR_I64
*
* *****
* * Structure for PT_Read_U. *
* *****
*
PtBR_U          DSECT
PtBR_U_NumOffsets DS    F      Number of entries in offset/control
                             word array
                             Reserved
PtBR_U_OWArray  DS    CL4      Array of offsets and control words
PtBR_U_Offset   DS    F      Offset of fullword of control
                             information
PtBR_U_Word     DS    F      fullword of control information
                             from user area in the debugged
                             process
PtBR_U_Len      EQU    *-PtBR_U

```

## BPXYPTXL — Map the parameter list for pthread\_create

AMODE 64 callers use “BPXYPTXL — Map the parameter list for pthread\_create” on page 1208.

```

BPXYPTXL ,
** BPXYPTXL: Pthread Parameter List
** Used By: PTX
PTXL          DSECT ,      Parameter List returned by BPX1PTX
PTXLWORKAREPTR DS    A      Pointer to User Work Area
PTXLATTRIBTEPTR DS    A      Pointer to User Attributes
PTXLTHIDPTR   DS    A      Pointer to Thread ID
PTXLSTATUSPTR DS    A      Pointer to Thread Run Status
PTXL#LENGTH   EQU    *-PTXL
PTXLRS        DSECT ,      Thread Run Status
PTXLRS        DS    0F
PTXLRSFLAGS   DS    0BL4   Thread Run Status Flags
PTXLRSFLAGS0  DS    B      1st byte
PTXLRSREADY   EQU    X'80'  Thread is ready to run
PTXLRSFLAGS1  DS    B      2nd byte
PTXLRSFLAGS2  DS    B      3rd byte
PTXLRSFLAGS3  DS    B      4th byte
PTXLRS#LENGTH EQU    *-PTXLRS
** BPXYPTXL End

```

## BPXYRFIS — Map the register file interest structures

```

BPXYRFIS ,
** BPXYRFIS
*
* Register File Interest Structure
*
RFIS          DSECT ,
RFIS_CMD      DS    H      CMD = REG OR UNREG
RFIS_FLAGS    DS    0H     FLAGS
RFIS_FLAGS1   DS    XL1    FLAGS Byte 1
RFIS_LOSTMSG  EQU    X'80'  MSGSND HAS FAILED
RFIS_FLAGS2   DS    XL1    FLAGS Byte 2

```

```

RFIS_RFTOK      DS    CL8      SYSTEM REGISTERED FILE TOKEN
RFIS_QID        DS    F        IPC MSG QUEUE ID
RFIS_TYPE       DS    F        IPC MSG TYPE
RFIS_UTOK       DS    CL8      IPC MSG USER TOKEN
*
RFIS#LENGTH     EQU    *-RFIS Length of this structure
*
** RFIS_CMD Values
RFIS#REG        EQU    1
RFIS#UNREG      EQU    2
*
*   Registered File invalidate Message
*
RFIM            DSECT ,
RFIM_TYPE       DS    F        IPC MSG TYPE, FROM RFIS_TYPE
RFIM_TEXT       DS    0CL12    IPC MSG TEXT:
RFIM_UTOK       DS    CL8      USER TOKEN, FROM RFIS_UTOK
RFIM_EVENT      DS    H        CHANGE EVENT
RFIM_FLAGS      DS    0H       FLAGS
RFIM_FLAGS1     DS    XL1      FLAGS Byte 1
RFIM_LOSTMSGSGS EQU    X'80'    MSGSND HAS FAILED
RFIM_FLAGS2     DS    XL1      FLAGS Byte 2
*
RFIM#LENGTH     EQU    *-RFIM Length of this structure
*
** RFIM_EVENT Values
RFIM#WRITE      EQU    1        WRITE, TRUNC, OPEN(O_TRUNC)
RFIM#ATTR       EQU    2        ANY ATTR CHANGE, CHMOD, ETC.
RFIM#UNLINK     EQU    3        ANY NAME UNLINKED
RFIM#RENAME     EQU    4        ANY NAME RENAMED
RFIM#UNMOUNT    EQU    5        CONTAINING FILE SYS UNMNTD
*
** BPXYRFIS End

```

## BPXYRLIM — Map the rlimit, rusage, and timeval structures

AMODE 64 callers use “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1208.

```

                BPXYRLIM ,
** BPXYRLIM: Rlimit, Timeval, and Rusage Structures
** Used By: setrlimit, getrlimit, and getrusage
RLIMIT          DSECT ,      Rlimit structure
RLIM_CUR_DW     DS    0CL8    Current limit (doubleword)
RLIM_CUR_HW     DS    F      Current (soft) limit highword -      X
                                used only for RLIMIT_FSIZE          X
                                and RLIMIT_MEMLIMIT, it is         X
                                ignored for all other resources
RLIM_CUR        DS    0F      Current (soft) limit lowword
RLIM_CUR_LW     DS    F      Current (soft) limit lowword
RLIM_MAX_DW     DS    0CL8    Current limit (doubleword)
RLIM_MAX_HW     DS    F      Current (hard) limit highword -      X
                                used only for RLIMIT_FSIZE          X
                                and RLIMIT_MEMLIMIT, it is         X
                                ignored for all other resources
RLIM_MAX        DS    0F      Maximum (hard) limit lowword
RLIM_MAX_LW     DS    F      Maximum (hard) limit lowword
RLIMIT#LENGTH   EQU    *-RLIMIT Length of this DSECT
TIMEVAL         DSECT ,      Timeval structure
TMVL_SEC        DS    F      Seconds
TMVL_USEC       DS    F      Microseconds
TIMEVAL#LENGTH  EQU    *-TIMEVAL Length of this DSECT
RUSAGE          DSECT ,      Rusage structure
RU_UTIME        DS    CL(TIMEVAL#LENGTH) User time used
RU_STIME        DS    CL(TIMEVAL#LENGTH) System time used
RUSAGE#LENGTH   EQU    *-RUSAGE Length of this DSECT
** BPXYRLIM End

```

## BPXYRMON — Map resource monitor data

```

                BPXYRMON ,
** BPXYRMON: Resource monitor data mapping
** Used By: RMG

```

RMON	DSECT	,	
RMONID	DC	C'RMON'	Eye catcher
RMONLENGTH	DC	A(RMON#LENGTH)	Length of this structure
RMONSYSCALLS	DS	F	Total Syscalls. This
*			includes syscalls done internally
*			by the kernel. It does not include
*			all trivial syscalls.
RMONCPU TIME	DS	F	Total CPU time spent in
*			kernel (Hundredths of a second)
RMONOVERRUN	DS	0CL12	
RMONOVRPROC	DS	F	Count of times the maximum number
*			of processes was exceeded.
RMONOVRUID	DS	F	Count of times the maximum number
*			of active UIDs was exceeded.
RMONOVRPRUID	DS	F	Count of times the maximum number
*			of processes per UID was exceeded.
RMONLIMITS	DS	0CL6	
RMONMAXPROC	DS	H	Maximum number of processes
RMONMAXUID	DS	H	Maximum number of active UIDs
RMONMAXPRUID	DS	H	Maximum number of processes per UID
RMONCURRENT	DS	0CL6	
RMONNUMPROC	DS	H	Current number of processes
RMONNUMUID	DS	H	Current number of active UIDs
	DS	H	Reserved
RMONOVERRUNIPC	DS	0CL16	
RMONOVRIPCMSGNIDS	DS	F	Number of attempts to exceed
*			maximum number of message queue
*			IDs
RMONOVRIPCSEMNIDS	DS	F	Number of attempts to exceed
*			maximum number of semaphore
*			IDs
RMONOVRIPCshmNIDS	DS	F	Number of attempts to exceed
*			maximum number of shared memory
*			IDs
RMONOVRIPCshmSPGS	DS	F	Number of attempts to exceed
*			maximum number of shared memory
*			pages for all segments
RMONLIMITSIPC	DS	0CL16	
RMONMAXIPCMSGNIDS	DS	F	Maximum number of message queue
*			IDs
RMONMAXIPCSEMNIDS	DS	F	Maximum number of semaphore
*			IDs
RMONMAXIPCshmNIDS	DS	F	Maximum number of shared memory
*			IDs
RMONMAXIPCshmSPGS	DS	F	Maximum number of shared memory
*			pages for all segments
RMONCURRENTIPC	DS	0CL16	
RMONNUMIPCMSGNIDS	DS	F	Current number of message queue
*			IDs
RMONNUMIPCSEMNIDS	DS	F	Current number of semaphore
*			IDs
RMONNUMIPCshmNIDS	DS	F	Current number of shared memory
*			IDs
RMONNUMIPCshmSPGS	DS	F	Current number of shared memory
*			pages for all segments
RMONOVRMMAPAREA	DS	F	Number of attempts to exceed
*			maximum number of mmap storage
*			pages
RMONMAXMMAPAREA	DS	F	Maximum number of mmap storage
*			pages
RMONNUMMMAPPAGES	DS	F	Current number of mmap storage
*			pages (in use)
RMONNUMSHRPAGES	DS	F	Current number of shared storage
*			pages
RMONOVRSHRPAGES	DS	F	Number of attempts to exceed
*			maximum number of shared storage
*			pages
RMONMAXSHRLIBRGN	DS	F	Maximum amount of storage available
*			for shared library region as
*			specified by parmlib statement
*			SHRLIBRGNSIZE in megabytes
RMONCURSHRLIBRGN	DS	F	Current amount of shared library
*			storage allocated in megabytes
RMONOVRSHRLIBRGN	DS	F	Number of attempts to exceed maximum
*			storage amount for shared library
*			region
RMONMAXQUEUEDSIGs	DS	F	Maximum amount of queued signals
*			allowed per process as specified
*			by parmlib statement
*			MAXQUEUEDSIGs
RMONOVRQUEUEDSIGs	DS	F	Number of attempts to exceed maximum
*			number of queued signals

```

RMON#LENGTH      EQU    *-RMON Length of RMON
** BPXYRMON End

```

## BPXYSECI — Map the output of BPX1IOC for the SECIGET request

```

          BPXYSECI      ,
** BPXYSECI: Socket Peer Security Identifiers
** Used By: IOC
SECI      DSECT      ,
SECIUSERID DS    CL8    MVS User ID
SECIEUID   DS    F      Effective UID
SECIEGID   DS    F      Effective GID
SECI#LENGTH EQU    *-SECI Length of this area
** BPXYSECI End

```

## BPXYSECO — Map the input/output of BPX1IOC for the SIOCSECENVR request

```

          BPXYSECO      ,
** BPXYSECO: Security Environment Object
** Used By: IOC
SECO      DSECT      ,
SECO_ARGUMENT DS    F      Input: SET / GET argument.
SECO_ENVR_OBJECT DS    0CL14 GET Output: Security ENVR OBJECT:
SECO_OBJECTLEN DS    F      GET Output: ENVR Object length.
SECO_BUFFERLEN DS    F      GET Input/Output: Buffer Length.
SECO_BUFFERADDR DS    A      GET Input/Output: Buffer Address.
SECO_BUFFERSP DS    X      GET Input/Output: Buffer SubPool.
SECO_BUFFERKEY DS    X      GET Input/Output: Buffer Key.
SECO#LENGTH EQU    *-SECO Length of this area
** BPXYSECO End

```

## BPXYSECT — Map the output of BPX1IOC for the SECIGET\_T request

```

          BPXYSECT      ,
** BPXYSECT: Socket Peer Security Identifiers
** Used By: IOC
SECT      DSECT      ,
SECTPUSERID DS    CL8    Process MVS User ID
SECTPEUID   DS    F      Process Effective UID
SECTPEGID   DS    F      Process Effective GID
SECTPUSERIDLEN DS    F      Process MVS User ID Length
SECTTUSERID DS    CL8    Task MVS User ID
SECTTEUID   DS    F      Task Effective UID
SECTTEGID   DS    F      Task Effective GID
SECTTUSERIDLEN DS    F      Task MVS User ID Length
SECT#LENGTH EQU    *-SECT Length of this area
** BPXYSECT End

```

## BPXYSEEK — Constants for lseek

BPXYSEEK is composed only of EQUates. DSECT= is allowed but ignored.

```

          BPXYSEEK      ,
** BPXYSEEK: Lseek constant definitions
** Used By: LSK
SEEK_SET    EQU    0      Set file offset to offset
SEEK_CUR    EQU    1      Set file offset to current + offset
SEEK_END    EQU    2      Set file offset to EOF + offset

```

```
** BPXYSEEK End
```

## BPXYSEL – Map the select options

BPXYSEL contains the read, write and exception options for the select system call.

```

                BPXYSEL
** BPXYSEL: Select Options
** Used By: SEL
SEL             DSECT ,
SELBEGIN       DS      0F
*
SELBITS        DS      0XL4   Flag Bits.8F FF FF FF Reserved
SELPOLLFLAGS   DS      XL2    Select flags / Poll (r)events
*-----
* Select flags
*-----
SELFLAGS       ORG      SELPOLLFLAGS
                DS      XL1
*
SELREAD        EQU      X'80'   Never use this bit
SELWRITE       EQU      X'40'   Descriptor ready for read.
SELXCEPT     EQU      X'20'   Descriptor ready for write.
                EQU      X'10'   Descriptor ready for exception.
                DS      XL1     Available byte
*-----
* Poll Events/Returned Events
*-----
SELPOLLEVENTS  ORG      SELPOLLFLAGS
                DS      XL2     Mapped by PollEvents(BPXYPOLL)
SELPOLLREVENTS ORG      SELPOLLFLAGS
                DS      XL2     Mapped by PollRevents(BPXYPOLL)
*
                DS      XL1     Available byte
                DS      XL1     Reserved for internal use
*
* Constants
*
SEL#LENGTH     EQU      *-SEL   Length of SEL
SEL#QUERY      EQU      1       Query function
SEL#CANCEL     EQU      2       Cancel function
SEL#BATSELQ    EQU      3       Batch-Select Query function
SEL#BATSELC    EQU      4       Batch-Select Cancel function
SEL#POLLQUERY  EQU      5       Poll Query function
SEL#BATPOLLQ   EQU      6       Batch-Poll Query function
SEL#BATPOLLC   EQU      7       Batch-Poll Cancel function
SEL#POLLCANCEL EQU      8       Poll Cancel function
SEL#BITSBACKWARD EQU      0     Bit Backward Order by word
SEL#BITSFORWARD EQU      1     Bit Forward Order by word
SEL#TYPES      EQU      3       3 TYPES (Read Write Except)
SEL#RBIT       EQU      64      Read bit position in byte
SEL#WBIT       EQU      32      Write bit position in byte
SEL#XBIT       EQU      16      Xcept bit position in byte
** BPXYSEL End

```

## BPXYSELT – Map the timeout value for the select syscall

AMODE 64 callers use “BPXYSELT – Map the timeout value for the select syscall” on page 1209.

```

                BPXYSELT
** BPXYSELT: Select Time Structure
** Used By: Select Syscall
SELT           DSECT ,
SELTBEGIN     DS      0D
*-----
*-----31-bit format
*
TV_SEC         DS      F'0'    Seconds
TV_USEC        DS      F'0'    Microseconds
* Constants
*
SELT#LENGTH    EQU      *-SELT Length of SELT

```

```
** BPXYSELT End
```

## BPXYSEM – Map interprocess communication semaphores

DSECTs (SEMID\_DS, SEM\_ARRAY and SEM\_BUF\_ELE) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT.

AMODE 64 callers use “BPXYSEM – Map interprocess communication semaphores” on page 1209.

```

                BPXYSEM
** BPXYSEM: Interprocess Communications Permission
** Used By: XSO, XSC
SEMID_DS       DSECT ,      semctl structure
SEM_PERM       DS      CL(IPC#LENGTH)  Mapped by BPXYIPCP
SEM_NSEMS      DS      H      number of semaphores in set
               DS      H      spacer
SEM_otime      DS      FL4      last semop() time
SEM_ctime      DS      FL4      last time changed by semctl()
SEM#LENGTH     EQU      *-SEMID_DS  Length of this DSECT
* SETVAL - a one element array for Semaphore_Number
* SETALL, GETALL - an array with Number_of_Semaphore elements
SEM_ARRAY      DSECT ,      SETALL, GETALL, SETVAL
SEM_ARRAY_VAL  DS      FL2      semaphore value
SEM_BUF_ELE    DSECT ,      sembuf element - semop
SEM_NUM        DS      FL2      semaphore number (0 to n-1)
SEM_OP         DS      FL2      semaphore operation
SEM_FLG        DS      H      operation flags
SEM#BUFLEN     EQU      *-SEM_BUF_ELE
* Flag bits - semop (also IPC_NOWAIT
SEM_UNDO       EQU      2      Set up adjust on exit entry.
* Control Commands - (also IPC_RMID, IPC_SET, IPC_STAT):
SEM_GETVAL     EQU      21     Get the current semaphore value
SEM_SETVAL     EQU      22     Change the semaphore value
SEM_GETPID     EQU      23     Get PID of last process to alter sem
SEM_GETNCNT    EQU      24     Get count of tasks waiting for val>0
SEM_GETZCNT    EQU      25     Get count of tasks waiting for val=0
SEM_GETALL     EQU      26     Get the current semaphore values
SEM_SETALL     EQU      27     Change the semaphore values
* Maximum and minimum values
SEM#MAX_VAL    EQU      32767   Maximum sem_val (min = 0)
SEM#MAX_ADJ    EQU      16383   Maximum sem_adj (min = -MAX)
** BPXYSEM End

```

## BPXYSFDL – Map the server file descriptor list structure

The mapping macro only provides enough space for one file descriptor; follow the invocation with up to 63 additional words.

```

                BPXYSFDL
** BPXYSFDL: Dile descriptor List
** Used By: SPW
SFDL           DSECT ,
SFDLHEADER     DS      0CL8
SFDLCOUNT      DS      F      Number of entries in this file descriptor list
SFDLFLAGS      DS      F      Flags
SFDLCLOSE      EQU      X'80'  All files to be closed (Bit 0 of SFDLFLAGS)
SFDLDESC       DS      F      First FD(follow by COUNT-1 additional FDs)
SFDLMAXCOUNT  EQU      64     Maximum value for SFDLCOUNT
SFDL_LEN       EQU      *-SFDL
** BPXYSFDL End

```



## BPXYSFPL — Map the send\_file parameter list

AMODE 64 callers use “BPXYSFPL — Map the send\_file parameter list” on page 1210.

```

                BPXYSFPL
** BPXYSFPL: SFPL system call structure
** Used By: BPX1SF
SFPL           DSECT ,
SFsocketDES    DS    F      Socket Descriptor
SFheaderLEN    DS    F      Header Length
SFheaderVPTR   DS    0F
SFheaderALET   DS    F      Header Alet
SFheaderPTR    DS    F      31-bit Header Ptr
SFfileDES      DS    F      File Descriptor
SFfileBYTESDW  DS    0F     Bytes to send Double Word (-1=all)
SFfileBYTESH   DS    F      High Word
SFfileBYTESL   DS    F      Low Word
SFfileOFFSETDW DS    0F     Offset Double Word
SFfileOFFSETH  DS    F      High Word
SFfileOFFSETL  DS    F      Low Word
SFfileSIZEDW   DS    0F     File Size Double Word
SFfileSIZEH    DS    F      High Word
SFfileSIZEL    DS    F      Low Word
SFtrailerLEN   DS    F      Trailer Length
SFtrailerVPTR  DS    0F
SFtrailerALET  DS    F      Trailer Alet
SFtrailerPTR   DS    F      31-bit Trailer Ptr
SFbytesSENTDW  DS    0F     Bytes Sent Double Word
SFbytesSENTH   DS    F      High Word
SFbytesSENTL   DS    F      Low Word
SFflags        DS    0XL4   Control Flags
SFPLVERSION    DS    XL1    Version
SFFLAGBYTE2    DS    XL1    Reserved
SFFLAGBYTE3    DS    XL1    Reserved
SFFLAGBYTE4    DS    XL1    Flags
SF_CLOSE       EQU    2     Close Socket Descriptor
SF_REUSE       EQU    1     Reuse Socket Descriptor
SFRESERVE      DS    CL12   Reserved
*
SFPLEND        EQU    *
*
SFPL#LENGTH    EQU    SFPLEND-SFPL
*
* Constants
*
** BPXYSFPL End

```

## BPXYSHM—Map interprocess communication shared memory segments

AMODE 64 callers use “BPXYSHM—Map interprocess communication shared memory segments” on page 1210.

```

                BPXYSHM
** BPXYSHM: Interprocess Communications Permission
** Used By: XMC
SHMID_DS       DSECT ,      SHMID_DS - shmctl structure
SHM_PERM       DS    CL(IPC#LENGTH) Mapped by BPXYIPC
SHM_SEGSZ      DS    F      size of segment in bytes
SHM_LPID       DS    F      process ID of last operation
SHM_CPID       DS    F      process ID of creator
SHM_NATTCH     DS    F      number of current attaches
SHM_ETIME      DS    F      time of last shmat
SHM_DTIME      DS    F      time of last shmdt
SHM_CTIME      DS    F      time of last change shmget/shmctl
* Mode bits (mapped over S_TYPE in BPXYMODE):
SHM_RDONLY     EQU    1     Attach read-only (else read-write)
SHM_RND        EQU    2     Round attach address to SHMLBA
SHMLBA         EQU    4096  Rounding boundary
SHM#LENGTH     EQU    *-SHMID_DS Length of this DSECT

```

```
** BPXYSHM End
```

## BPXYSIGH – Signal constants

BPXYSIGH is composed of only EQUates. DSECT= is allowed but ignored.

```

                BPXYSIGH
** BPXYSIGH: Component signal definition
** Used By: KIL SIA SPM
*****
* Signals with default action ABNORMAL TERMINATION
SIGHUP# EQU 1 Hangup detected on controlling terminal
SIGINT# EQU 2 Interactive attention
SIGABRT# EQU 3 Abnormal termination
SIGILL# EQU 4 Detection of an incorrect hardware instruction
SIGPOLL# EQU 5 Pollable event
SIGURG# EQU 6 High bandwidth data is available at a socket
SIGFPE# EQU 8 Erroneous arithmetic operation, such as division
* by zero of an operation resulting in overflow
SIGKILL# EQU 9 Termination (cannot be caught or ignored)
SIGBUS# EQU 10 Bus error
SIGSEGV# EQU 11 Detection of an incorrect memory reference
SIGSYS# EQU 12 Bad System Call
SIGPIPE# EQU 13 Write on a pipe with no readers
SIGALRM# EQU 14 Timeout
SIGTERM# EQU 15 Termination
SIGUSR1# EQU 16 Reserved as application-defined signal 1
SIGUSR2# EQU 17 Reserved as application-defined signal 2
SIGABND# EQU 18 Abend
SIGQUIT# EQU 24 Interactive termination
SIGTRAP# EQU 26 Trap used by the ptrace call
SIGXCPU# EQU 29 CPU time limit exceeded
SIGXFSZ# EQU 30 File size limit exceeded
SIGVTALRM# EQU 31 Virtual timer expired
SIGPROF# EQU 32 Profiling timer expired
SIGDANGER# EQU 33 Shutdown Imminent
* Signals with default action of CONTINUE
* Signals with default action IGNORE THE SIGNAL
SIGNULL# EQU 0 Null - no signal sent
SIGCHLD# EQU 20 Child process terminated or stopped
SIGIO# EQU 23 Completion of input or output
SIGIOER# EQU 27 Input or Output Error
SIGWINCH# EQU 28 Change size of window
SIGTRACE# EQU 37 Trace the target process
SIGDUMP# EQU 39 Take a SYSMDUMP
* Signals with default action STOP
SIGSTOP# EQU 7 Stop (cannot be caught or ignored)
SIGTTIN# EQU 21 Read from a control terminal attempted by a
* member of a background process group
SIGTTOU# EQU 22 Write from a control terminal attempted by a
* member of a background process group
SIGTSTP# EQU 25 Interactive stop
SIGTHSTOP# EQU 34 Thread stop (cannot be caught or blocked or
* ignored)
* Signals with default action CONTINUE IF IT IS CURRENTLY STOPPED,
* OTHERWISE IGNORE THE SIGNAL
SIGCONT# EQU 19 Continue if stopped
SIGTHCONT# EQU 35 Thread continue (cannot be caught or blocked or
* ignored)
*****
** Equates that define sa_handler values on Sigaction()
*****
SIG_DFL# EQU 0 Default signal action
SIG_IGN# EQU 1 Ignore signal action
*****
** Constants that define sa_flags values on Sigaction()
*****
SA_FLAGS_DFT# EQU X'00000000' Default sa_flags
SA_NOCLDSTOP# EQU X'80000000' No SIGCHLD when children stop
SA_OLD_STYLE# EQU X'40000000' Old style signal() function
SA_ONSTACK# EQU X'20000000' Deliver on alternate stack
SA_RESETHAND# EQU X'10000000' Reset action on delivery
SA_RESTART# EQU X'08000000' Restart interruptible funcs
SA_SIGINFO# EQU X'04000000' Pass siginfo to catcher
SA_NOCLDWAIT# EQU X'02000000' Don't create zombie on exit
SA_NODEFER# EQU X'01000000' Don't block signal on delivery
SA_IGNORE# EQU X'00000001' Act as though sa_handler contained

```

```

*                               SIG_IGN#
*****
** Constants that define how parameter on sigprocmask()
*****
SIG_BLOCK#      EQU    0   Block signals set on in New_signal_mask
SIG_UNBLOCK#    EQU    1   Unblock signals set on in New_signal_mask
SIG_SETMASK#    EQU    2   Set signal mask to New_signal_mask
*****
** Constants that define the lower two bytes of the Signal_Options *
** on the BPX1KIL and BPX1PTK syscalls.  If a signal generated with *
** one or more of these flags is handled by the Signal Interface *
** Routine, the flags will appear in the PpsdKilOpts field upon *
** delivery of said signal.
** When the lower two bytes contain x'1000' (SIG_CONSCANCEL#) the *
** upper two bytes will contain the SIGCNCL type qualifier
*****
SIG_FLAGS_DFT#   EQU    X'0000' Default options
SIG_PTRACEBYPASS# EQU    X'8000' Bypass ptrace processing
SIG_KERNSICODE#  EQU    X'4000' z/OS UNIX kernel set si_code
SIG_APPLSICODE#  EQU    X'2000' Application set si_code
SIG_CONSCANCEL#  EQU    X'1000' Console (MODIFY) cancel thread
*****
** Constants that define si_codes which are passed in the upper two *
** bytes of the Signal_Options on the BPX1KIL and BPX1PTK syscalls *
** If a signal generated with a si_code is handled by the Signal *
** Interface Routine the si_code will appear in the PpsdKilData *
** field upon delivery of said signal.
*****
ILL_ILLOPC#      EQU    11   Illegal opcode
ILL_ILLOPN#      EQU    12   Illegal operand
ILL_ILLADR#      EQU    13   Illegal addressing mode
ILL_ILLTRP#      EQU    14   Illegal trap
ILL_PRVOPC#      EQU    15   Privileged opcode
ILL_PRVREG#      EQU    16   Privileged register
ILL_COPROC#      EQU    17   Coprocessor error
ILL_BADSTK#      EQU    18   Internal stack error
ILL_EXECUTE#     EQU    19   Execute exception
ILL_ILLSPEC#     EQU    20   Specification exception
*****
FPE_INTDIV#      EQU    31   Integer divide by zero
FPE_INTOVF#      EQU    32   Integer overflow
FPE_FLTDIV#      EQU    33   Floating point divide by zero
FPE_FLTOVF#      EQU    34   Floating point overflow
FPE_FLTUND#      EQU    35   Floating point underflow
FPE_FLTRES#      EQU    36   Floating point inexact result
FPE_FLTINV#      EQU    37   Invalid floating point operation
FPE_FLTSUB#      EQU    38   Subscript out of range
FPE_FLTSIG#      EQU    39   Floating point significance exception
FPE_DECDATA#     EQU    40   Decimal data exception
FPE_DECDIV#      EQU    41   Decimal divide by zero
FPE_DECOVF#      EQU    42   Decimal overflow
FPE_UNKWN#       EQU    43   Undetermined exception
*****
SEGV_MAPERR#     EQU    51   Address not mapped to object
SEGV_ACCERR#     EQU    52   Invalid permissions for mapped object
SEGV_PROTECT#    EQU    53   Invalid key access
SEGV_ADDRESS#    EQU    54   Invalid address
*****
BUS_ADRALN#      EQU    71   Invalid address alignment
BUS_ADRERR#      EQU    72   Non-existent physical address
BUS_OBJERR#      EQU    73   Object specific hardware error
*****
TRAP_BRKPT#      EQU    91   Process breakpoint
TRAP_TRACE#      EQU    92   Process trace trap
*****
CLD_EXITED#      EQU    101  Child has exited
CLD_KILLED#      EQU    102  Child was killed
CLD_DUMPED#      EQU    103  Child was terminated abnormally
CLD_TRAPPED#     EQU    104  Traced child has trapped
CLD_STOPPED#     EQU    105  Child has stopped
CLD_CONTINUED#   EQU    106  Stopped child was continued
*****
POLL_IN#         EQU    111  Data input available
POLL_OUT#        EQU    112  Output buffers available
POLL_MSG#        EQU    113  Input message available
POLL_ERR#        EQU    114  I/O error
POLL_PRI#        EQU    115  High priority input available
POLL_HUP#        EQU    116  Device disconnected
*****
ABND_REAL#       EQU    170  Abend Real
*****
SI_ASYNCIO#      EQU    175  Completion of an asynchronous I/O

```

```

SI_QUEUE#      EQU 176  Signal sent by sigqueue()
*****
** Equate for BPX1STW (sigtimedwait) syscall that when specified  *
** for the "Seconds" parameter indicates not to timeout while      *
** waiting for signal(s).                                         *
*****
SIG#NO_TIMEOUT EQU X'7FFFFFFF'
** BPXYSIGH End

```

## BPXYSINF — Map SIGINFO\_T structure

DSECT (SIGINFO\_T) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT.

AMODE 64 callers use “BPXYSINF — Map SIGINFO\_T structure” on page 1211.

```

          BPXYSINF
** BPXYSINF: siginfo_t Structure
** Used By: waitid
SIGINFO_T DSECT ,      Siginfo_t structure
SI_SIGNO  DS  F      signal number
SI_ERRNO  DS  F      error number
SI_CODE   DS  F      signal code
SI_PID    DS  F      sending process ID
SI_UID    DS  F      real user ID of sending process
SI_ADDR   DS  A      address of faulting instruction
SI_STATUS DS  F      exit value or signal
SI_BAND   DS  F      band event for SIGPOLL
SI_VALUE  DS  F      signal value
SIGINFO#LENGTH EQU *-SIGINFO_T Length of this DSECT
** BPXYSINF End

```

## BPXYSMC — Map shared mutex/condvar declares and constants

BPXYSMC maps declares and constants for shared mutex/condvar support

```

          BPXYSMC ,
*
* *****
* * Define equates for FcnCode parameter *
* *****
*
*
SMC_WAIT EQU 1      Wait function request
SMC_POST EQU 2      Post function request
SMC_INIT EQU 4      Initialization function request
SMC_DESTROY EQU 8    Destroy function request
SMC_POSTALL EQU 16   Postall function request
SMC_SETUPWAIT EQU 32 SetupToWait function request
SMC_CANCELSETUPWAIT EQU 64 CancelSetupToWait function request
*
* *****
* * Mapping for FcnFlags parameter *
* *****
*
*
SMC_FCNFLAGS DSECT
SMC_FCNFLAGSB1 DS 0BL1
SMC_MUTEX EQU X'80'
SMC_CONDVAR EQU X'40'
SMC_TIMEDWAIT EQU X'20'
SMC_OUTSIDEWAIT EQU X'10'
          ORG SMC_FCNFLAGSB1+X'00000001'
SMC_FCNFLAGSB2 DS 1BL1
SMC_FCNFLAGSB3 DS 1BL1
SMC_FCNFLAGSB4 DS 1BL1
SMC_FCNFLAGS_LEN EQU *-SMC_FCNFLAGS
*
* *****
* * Mapping for Time Structure pointed to by TimeStrucAddr parameter *
* *****
*

```

```

*
SMCT      DSECT
SMCTSECS DS    1FL4      The time to wait for the condition variable   X
                        expressed in seconds. Seconds can be any   X
                        value greater or equal to 0 or less than or X
                        equal to 4,294,967,295.
SMCTNANOSECS DS 1FL4      The time in nanoseconds to be added to   X
                        SmctSecs to wait for condition variable.   X
                        Nanoseconds can be any value greater than or X
                        equal to 0 and less than 1,000,000,000.
SMCT_LEN EQU    *-SMCT

```

## BPXYSOCK — Map SOCKADDR structure and constants

BPXYSOCK maps the SOCKADDR structure for socket, accept, bind, sendto, recvfrom, getsockname, and getpeername.

The SOCK\_LEN field is required for AF\_UNIX, but is optional for AF\_INET and AF\_INET6. Specifically for AF\_INET6, SOCK\_LEN can either be defined as zero or SOCK#LEN+SOCK\_SIN6#LEN.

```

                BPXYSOCK      ,
*
*****
** BPXYSOCK: z/OS UNIX Socket Address Structure
** Used By: Sockets PFS
*****
*
SOCKADDR      DSECT ,
                AGO .C411
                ANOP ,
SOCKADDR      DS    0F
                .C411 ANOP ,
SOCKBEGIN     DS    0F
*
SOCK_LEN      DS    X      Address Length - Length of
                        either SOCK_SIN (for AF_INET
                        sockets) or of the name supplied**
                        in SOCK_SUN_NAME (for AF_UNIX
                        sockets)
SOCK_FAMILY   DS    X      Address Family
SOCK_DATA     DS    0C      Protocol specific area
*
SOCK#LEN      EQU    *-SOCKADDR Constant - Fixed length of SOCK
*
*****
*
*   AF_Inet Socket Address Structure
*
*****
*
                ORG  SOCK_DATA   Start of AF_Inet unique area
SOCK_SIN      DS    0C
SOCK_SIN_PORT DS    H           Port number used by the appl
SOCK_SIN_ADDR DS    CL4        INET address (netid)
                DS    CL8        Reserved area not used
*
SOCK_SIN#LEN  EQU    *-SOCK_SIN Constant - Fixed length of
                        AF_Inet unique area
*
*
*****
*
*   AF_UNIX Socket Address Structure
*
*****
*
                ORG  SOCK_DATA   Start of AF_Unix unique area
SOCK_SUN      DS    0C
SOCK_SUN_NAME DS    CL108       Path name of the socket @P0C
                Length 108 matchs RS/6000@P0A
*
SOCK_SUN#LEN  EQU    *-SOCK_SUN Constant - Fixed length of
                        AF_Unix unique area
*
*
*****
*
*   AF_Inet6 Socket Address Structure
*
*****

```

```

*
*****
*
SOCK_SIN6          ORG  SOCK_DATA  Start of AF_Inet6 area      @PBA
SOCK_SIN6          DS    0C          @PBA
SOCK_SIN6_PORT     DS    H          Port number used by the appl @PBA
SOCK_SIN6_FLOWINFO DS    CL4        FLOW INFORMATION         @PBA
SOCK_SIN6_ADDR     DS    CL16       INET address (netid)      @PBA
SOCK_SIN6_SCOPE_ID DS    CL4        SCOPE ID                 @PBA
*
SOCK_SIN6#LEN      EQU  *-SOCK_SIN6 Length of AF_INET6 area  @PBA
*
*****
* Equates for Address Families
*
*****
*
AF_UNSPEC          EQU  0          Unspecified
AF_UNIX            EQU  1          Unix Domain
AF_INET            EQU  2          Internetwork: UDP TCP
AF_IMPLINK         EQU  3          Arpanet imp addresses
AF_PUP             EQU  4          pup protocols: BSP
AF_CHAOS           EQU  5          mit CHAOS protocols
AF_NS              EQU  6          XEROX NS protocols
AF_NBS             EQU  7          nbs protocols
AF_ECMA            EQU  8          European computer man.
AF_DATAKIT         EQU  9          datakit protocols
AF_CCITT           EQU  10         CCITT protocols: X.25
AF_SNA             EQU  11         IBM SNA
AF_DECNET          EQU  12         DECNet
AF_DLI             EQU  13         Direct data link interface
AF_LAT             EQU  14         LAT
AF_HYLINK          EQU  15         NSC hyperchannel
AF_APPLETALK       EQU  16         Apple Talk
AF_IUCV            EQU  17         IBM IUCV
AF_ESCON           EQU  18         ESCON UDP                  @D4A
AF_INET6           EQU  19         IPv6                        @P8A
AF_ROUTE           EQU  20         Routing Sockets             @P8A
AF_MAX             EQU  21
*
*****
* Equates for protocol
*
*****
*
IPPROTO_IP         EQU  0          DEFAULT PROTOCOL
IPPROTO_TCP        EQU  6          TCP
IPPROTO_UDP        EQU  17         USER DATAGRAM
IPPROTO_IPV6       EQU  41         IPv6                        @PBA
IPPROTO_ICMPV6     EQU  58         IPv6 ICMP                   @D9A
*
IPPROTO_HOPOPTS    EQU  0          @DAA
IPPROTO_ROUTING    EQU  43         @DAA
IPPROTO_FRAGMENT   EQU  44         @DAA
IPPROTO_ESP        EQU  50         @DAA
IPPROTO_AH         EQU  51         @DAA
IPPROTO_NONE       EQU  59         @DAA
IPPROTO_DSTOPTS    EQU  60         @DAA
*
*****
* Equates for setpeer options
*
*****
*
SOCK#SO_SET        DC    X'00000200'
SOCK#SO_SET_EQU    EQU    X'00000200'          @04A
SOCK#SO_UNSET      DC    X'00000400'
SOCK#SO_UNSET_EQU  EQU    X'00000400'          @04A
*
*****
* Equates for socket types
*
*****
*
SOCK#_STREAM       EQU  1
SOCK#_DGRAM        EQU  2
SOCK#_RAW          EQU  3
SOCK#_RDM          EQU  4
SOCK#_SEQPACKET    EQU  5
*

```

```

*****
*
*   Equates for Dimension      (socket/socketpair syscall)
*
*****
SOCK#DIM_SOCKET      EQU  1
SOCK#DIM_SOCKETPAIR  EQU  2
SOCK#DIM_SOCKETWAFFINITY EQU  3      @02A
SOCK#DIM_SOCKETPAIRWAFFINITY EQU  4      @02A
*
*****
*   Equates for getname option
*
*****
SOCK#GNMOPTGETPEERNAME EQU  1
SOCK#GNMOPTGETSOCKNAME EQU  2
*
*****
*   Equates for sockopt
*
*****
SOCK#OPTOPTGETSOCKOPT   EQU  1
SOCK#OPTOPTSETSOCKOPT   EQU  2
SOCK#OPTOPTSETIBMSOCKOPT EQU  3      @D5A
*
*****
*   Equates for Shutdown options
*
*****
SOCK#SHUTDOWNREAD      EQU  0
SOCK#SHUTDOWNWRITE      EQU  1
SOCK#SHUTDOWNBOTH      EQU  2
*
*****
*   Equate for Level Number for socket options
*
*****
SOCK#SOL_SOCKET      DC   X'0000FFFF'
SOCK#SOL_SOCKET_EQU  EQU  X'0000FFFF'      @04A
*
*****
*   Equate for InAddrAny for bind requests
*
*****
INADDR_ANY      DC   X'00000000'
INADDR_ANY_EQU  EQU  X'00000000'      @04A
*
INADDR_LOOPBACK DC   X'7F000001'      @PBA
INADDR_LOOPBACK_EQU EQU  X'7F000001'      @04A
IN6ADDR_ANY      DC   X'00000000000000000000000000000000' @PBA
IN6ADDR_LOOPBACK DC   X'00000000000000000000000000000001' @PBA
IN6ADDR_MAPPEDV4 DC   X'00000000000000000000FFFF'      @D9A
IN6ADDR_COMPATV4 DC   X'000000000000000000000000'      @D9A
*
*****
*   Equates for Socket options
*
*****
SOCK#SO_DEBUG      DC   X'00000001'
SOCK#SO_ACCEPTCONN DC   X'00000002'
SOCK#SO_REUSEADDR  DC   X'00000004'
SOCK#SO_KEEPAVIVE  DC   X'00000008'
SOCK#SO_DONTROUTE  DC   X'00000010'
SOCK#SO_BROADCAST  DC   X'00000020'
SOCK#SO_USELOOPBACK DC   X'00000040'
SOCK#SO_LINGER      DC   X'00000080'
SOCK#SO_OOBINLINE   DC   X'00000100'

```

```

SOCK#SO_REUSEPORT EQU X'00000200' To match socket.h @DHA
SOCK#SO_REUSEPORT2 EQU X'00000007' As implemented in TCPIP @DHA
*
SOCK#SO_SNDBUF DC X'00001001'
SOCK#SO_RCVBUF DC X'00001002'
SOCK#SO_SNDLOWAT DC X'00001003'
SOCK#SO_RCVLOWAT DC X'00001004'
SOCK#SO_SNDTIMEO DC X'00001005'
SOCK#SO_RCVTIMEO DC X'00001006'
SOCK#SO_ERROR DC X'00001007'
SOCK#SO_TYPE DC X'00001008'
*
* Non-standard sockopts
*
SO_PROPAGATEID DC X'00004000' @D7A
SO_CLUSTERCONNTYPE DC X'00004001' @P8A
SO_SECINFO DC X'00004002' @D9A
SO_RECVUSERNAME EQU X'00004003' @DHA
*
* EQUated Versions @04A
* @04A
SOCK#SO_DEBUG_EQU EQU X'00000001' @04A
SOCK#SO_ACCEPTCONN_EQU EQU X'00000002' @04A
SOCK#SO_REUSEADDR_EQU EQU X'00000004' @04A
SOCK#SO_KEEPAIVE_EQU EQU X'00000008' @04A
SOCK#SO_DONTROUTE_EQU EQU X'00000010' @04A
SOCK#SO_BROADCAST_EQU EQU X'00000020' @04A
SOCK#SO_USELOOPBACK_EQU EQU X'00000040' @04A
SOCK#SO_LINGER_EQU EQU X'00000080' @04A
SOCK#SO_OOBINLINE_EQU EQU X'00000100' @04A
SOCK#SO_SNDBUF_EQU EQU X'00001001' @04A
SOCK#SO_RCVBUF_EQU EQU X'00001002' @04A
SOCK#SO_SNDLOWAT_EQU EQU X'00001003' @04A
SOCK#SO_RCVLOWAT_EQU EQU X'00001004' @04A
SOCK#SO_SNDTIMEO_EQU EQU X'00001005' @04A
SOCK#SO_RCVTIMEO_EQU EQU X'00001006' @04A
SOCK#SO_ERROR_EQU EQU X'00001007' @04A
SOCK#SO_TYPE_EQU EQU X'00001008' @04A
SO_PROPAGATEID_EQU EQU X'00004000' @04A
SO_CLUSTERCONNTYPE_EQU EQU X'00004001' @04A
SO_SECINFO_EQU EQU X'00004002' @04A
*
* SO_CLUSTERCONNTYPE Output Values
*
SO_CLUSTERCONNTYPE_NOCONN EQU 0 @P8A
SO_CLUSTERCONNTYPE_NONE EQU 1 @P8A
SO_CLUSTERCONNTYPE_SAME_CLUSTER EQU 2 @P8A
SO_CLUSTERCONNTYPE_SAME_IMAGE EQU 4 @P8A
SO_CLUSTERCONNTYPE_INTERNAL EQU 8 @P8A
*
* IPPROTO_IP Options
*
IP_TOS EQU 2 @P9C@D6A
IP_MULTICAST_TTL EQU 3 @D6A
IP_MULTICAST_LOOP EQU 4 @D6A
IP_ADD_MEMBERSHIP EQU 5 @D6A
IP_DROP_MEMBERSHIP EQU 6 @D6A
IP_MULTICAST_IF EQU 7 @P9C@D6A
IP_DEFAULT_MULTICAST_TTL EQU 1 @D6A
IP_DEFAULT_MULTICAST_LOOP EQU 1 @D6A
IP_MAX_MEMBERSHIPS EQU 20 @D6A
IP_BLOCK_SOURCE EQU 10 @DEA
IP_UNBLOCK_SOURCE EQU 11 @DEA
IP_ADD_SOURCE_MEMBERSHIP EQU 12 @DEA
IP_DROP_SOURCE_MEMBERSHIP EQU 13 @DEA
*****
* Multicast Source Filter Structures from RFC 3678 *
***** @DEA
AIF ('&DSECT' EQ 'NO').B425 @DEA
SOCKADDR_STORAGE_STRUCT DSECT , @DEA
AGO .C425 @DEA
.B425 ANOP , @DEA
SOCKADDR_STORAGE_STRUCT DS 0D @DEA
.C425 ANOP , @DEA
SOCKADDR_STORAGE DS CL128 @DEA
*
* setibmssockopt options
*
SOCK#SO_BULKMODE DC X'00008000' @D5A
SOCK#SO_IGNOREINCOMINGPUSH DC X'00000001' @D5A
SOCK#SO_NONBLOCKLOCAL DC X'00008001' @P7A

```



```

SOCK#SO_IGNORESOURCEVIPA    DC X'00000002'          @P7A
*                               Toggles the use of non-VIPA addresses. When
*                               enabled, non-VIPA addresses will be used for
*                               outbound IP packets.
SOCK#SO_OPTMSS                DC X'00008003'          @P7A
*                               Toggles the use of optimal TCP segment size.
*                               When enabled, the TCP segment size may be optimally
*                               increased on outbound data transfers. This may
*                               reduce the amount of TCP outbound and inbound
*                               acknowledgement packet processing; therefore,
*                               minimizing CPU consumption.
SOCK#SO_OPTACK                DC X'00008004'          @P7A
*                               Optimize Acks
SOCK#SO_EIOIFNEWTP            DC X'00000005'          @PAA
*                               Notify of new tp
SOCK#SO_ACCEPTECONNABORTED    DC X'00000006'          @PEA
*                               Notify of conn abtd
SOCK#SO_EXCLWRT               DC X'00000007'          @03A
*                               Write Serialization
*                               Control Stream Write Serialization
*                               SetIbmSockOpt option to toggle system supplied
*                               serialization on TCP stream socket writes.
*                               NOTE: This function has been disabled and its @DIC
*                               use is thus discouraged. This function @DIC
*                               may be withdrawn in a future release.
*                               @DIC
*   EQUated Versions
SOCK#SO_BULKMODE_EQU          EQU X'00008000'          @04A
SOCK#SO_IGNOREINCOMINGPUSH_EQU EQU X'00000001'          @04A
SOCK#SO_NONBLOCKLOCAL_EQU     EQU X'00008001'          @04A
SOCK#SO_IGNORESOURCEVIPA_EQU  EQU X'00000002'          @04A
SOCK#SO_OPTMSS_EQU            EQU X'00008003'          @04A
SOCK#SO_OPTACK_EQU            EQU X'00008004'          @04A
SOCK#SO_EIOIFNEWTP_EQU        EQU X'00000005'          @04A
SOCK#SO_ACCEPTECONNABORTED_EQU EQU X'00000006'          @04A
SOCK#SO_EXCLWRT_EQU           EQU X'00000007'          @04A
*
*****
*   Equates for So_ option values
*
*****
SOCK#SO_SETOPTIONON           DC X'00000001'          @PAA
SOCK#SO_SETOPTIONON_EQU       EQU X'00000001'          @04A
SOCK#SO_SETOPTIONOFF          DC X'00000000'          @PAA
SOCK#SO_SETOPTIONOFF_EQU      EQU X'00000000'          @04A
*
*****
*   Equates for IPPROTO_TCP options
*
*****
SOCK#TCP_NODELAY              DC X'00000001'          @P4A
SOCK#TCP_NODELAY_EQU          EQU X'00000001'          @04A
SOCK#TCP_KEEPAALIVE           DC X'00000008'          @P9A
SOCK#TCP_KEEPAALIVE_EQU       EQU X'00000008'          @04A
*
*****
*   Equates for Socket Port Constant
*
*****
SOCK#LASTRESERVEPORT          EQU 1023                @D2A
*
*
*   AIF ('&DSECT' EQ 'NO').B412          @01A
IP_MREQ DSECT , @P9M@D6A
*   AGO .C412 , @01A
*   .B412 ANOP , @01A
IP_MREQ DS 0F @01A
*   .C412 ANOP , @01A
IMR_MULTIADDR DS CL4 IP MULTICAST ADDR OF GROUP @D6A
IMR_INTERFACE DS CL4 LOCAL IP ADDR OF INTERFACE @D6A
*
*****
*   IP_MREQ_SOURCE STRUCTURE
*
*****
*   AIF ('&DSECT' EQ 'NO').B422          @DEA
IP_MREQ_SOURCE DSECT , @DEA
*   AGO .C422 , @DEA
*   .B422 ANOP , @DEA
IP_MREQ_SOURCE DS 0F @DEA
*   .C422 ANOP , @DEA
IMRS_MULTIADDR DS CL4 IP MULTICAST ADDR @DEA
IMRS_SOURCEADDR DS CL4 IP SOURCE ADDR @DEA

```

```

IMRS_INTERFACE DS CL4 LOCAL IP ADDR OF INTERFACE @DEA
* @DEA
*****
* GROUP_REQ STRUCTURE
*
***** @DEA
AIF ('&DSECT' EQ 'NO').B423 @DEA
GROUP_REQ DSECT , @DEA
AGO .C423 @DEA
.B423 ANOP , @DEA
GROUP_REQ DS 0F @DEA
.C423 ANOP , @DEA
GR_INTERFACE DS CL4 INTERFACE INDEX @DEA
DS CL4 PADDING @DEA
GR_GROUP DS CL(L'SOCKADDR_STORAGE) GROUP ADDRESS @DEA
ORG GR_GROUP @DEA
GR_MULTISOCKADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN) @DEA
ORG GR_GROUP @DEA
GR_MULTISOCKADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN) @DEA
* @DEA
*****
* GROUP_SOURCE_REQ STRUCTURE
*
***** @DEA
AIF ('&DSECT' EQ 'NO').B424 @DEA
GROUP_SOURCE_REQ DSECT , @DEA
AGO .C424 @DEA
.B424 ANOP , @DEA
GROUP_SOURCE_REQ DS 0D @DEA
.C424 ANOP , @DEA
GSR_INTERFACE DS CL4 INTERFACE INDEX @DEA
DS CL4 PADDING @DEA
GSR_GROUP DS CL(L'SOCKADDR_STORAGE) GROUP ADDRESS @DEA
ORG GSR_GROUP @DEA
GSR_GROUPADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN) @DEA
ORG GSR_GROUP @DEA
GSR_GROUPADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN) @DEA
GSR_SOURCE DS CL(L'SOCKADDR_STORAGE) SOURCE ADDRESS @DEA
ORG GSR_SOURCE @DEA
GSR_SOURCEADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN) @DEA
ORG GSR_SOURCE @DEA
GSR_SOURCEADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN) @DEA
* @DEA
*****
* Structure for So_Linger
* @D3A*
*****
AIF ('&DSECT' EQ 'NO').B413 @01A
SOCK_LINGER_STRUCT DSECT , @P9A
AGO .C413 @01A
.B413 ANOP , @01A
SOCK_LINGER_STRUCT DS 0F @01A
.C413 ANOP , @01A
SOCK_L_ONOFF DS F On/Off indicator @P9M@D3A
SOCK_L_LINGER DS F Length of time to linger @P9M@D3C
*****
* Equates for IPPROTO_IPV6 Options
* @D9A *
*
***** @D9A
SOCK#IPV6_UNICAST_HOPS EQU 3
SOCK#IPV6_MULTICAST_LOOP EQU 4
SOCK#IPV6_JOIN_GROUP EQU 5
SOCK#IPV6_LEAVE_GROUP EQU 6
SOCK#IPV6_MULTICAST_IF EQU 7
SOCK#IPV6_MULTICAST_HOPS EQU 9
SOCK#IPV6_V6ONLY EQU 10
SOCK#IPV6_HOPLIMIT EQU 11 ANC DATA ONLY
SOCK#IPV6_PATHMTU EQU 12
SOCK#IPV6_PKTINFO EQU 13
SOCK#IPV6_RECVHOPLIMIT EQU 14
SOCK#IPV6_RECVPKTINFO EQU 15
SOCK#IPV6_RECVPATHMTU EQU 16
SOCK#IPV6_REACHCONF EQU 17
SOCK#IPV6_USE_MIN_MTU EQU 18
SOCK#IPV6_CHECKSUM EQU 19
SOCK#IPV6_NEXTHOP EQU 20

```

```

SOCK#IPV6_RTHDR EQU 21
SOCK#IPV6_HOPOPTS EQU 22
SOCK#IPV6_DSTOPTS EQU 23
SOCK#IPV6_RTHDRDSTOPTS EQU 24
SOCK#IPV6_RECVRTHDR EQU 25
SOCK#IPV6_RECVHOPOPTS EQU 26
*SOCK#IPV6_RECVRTHDRDSTOPTS EQU 27 @DCD
SOCK#IPV6_RECVDSTOPTS EQU 28
SOCK#IPV6_DONTFRAG EQU 29 @DCA
SOCK#IPV6_TCLASS EQU 30 @DCA
SOCK#IPV6_RECVTCLASS EQU 31 @DCA
SOCK#IPV6_ADDR_PREFERENCES EQU 32 @DJA

SOCK#IPV6_RTHDR_TYPE_0 EQU 0 IPv6 Routing hdr type 0 @D9A
*****
* Protocol Independent Options @DEA *
*****
SOCK#MCAST_JOIN_GROUP EQU 40 @DEA
SOCK#MCAST_LEAVE_GROUP EQU 41 @DEA
SOCK#MCAST_JOIN_SOURCE_GROUP EQU 42 @DEA
SOCK#MCAST_LEAVE_SOURCE_GROUP EQU 43 @DEA
SOCK#MCAST_BLOCK_SOURCE EQU 44 @DEA
SOCK#MCAST_UNBLOCK_SOURCE EQU 45 @DEA
*****
* Equates for IPPROTO_ICMPV6 options @D9A *
*
*****
SOCK#ICMP6_FILTER EQU 1

*****
* Structure for Packet Source/Destination Information @D9A *
*
*****
IN6_PKTINFO DSECT ,
    AGO .C414
    .B414 ANOP ,
IN6_PKTINFO DS 0F
    .C414 ANOP ,
IPI6_ADDR DS CL16 IPv6 Addr
IPI6_IFINDEX DS F Interface Index

*****
* Structure for Multicast Mreq @D9A *
*
*****
AIF ('&DSECT' EQ 'NO').B415
IPV6_MREQ DSECT ,
    AGO .C415
    .B415 ANOP ,
IPV6_MREQ DS 0F
    .C415 ANOP ,
IPV6MR_MULTIADDR DS CL16 IPv6 Addr
IPV6MR_INTERFACE DS F Interface index

*****
* Structure for CInet Interface Index @D9A *
*
*****
AIF ('&DSECT' EQ 'NO').B416
IFINDEX DSECT ,
    AGO .C416
    .B416 ANOP ,
IFINDEX DS 0F
    .C416 ANOP ,
IFI_TDX DS H Cinet Td Index
IFI_INDEX DS H Stacks Interface Index

*****
* Structure for Icmp6 Filtering @D9A *
*
*****
AIF ('&DSECT' EQ 'NO').B417
ICMP6_FILTER DSECT ,
    AGO .C417

```

```

.B417      ANOP      ,          DS      0F          @DBC
ICMP6_FILTER
.C417      ANOP      , ICMP6_FILT      DS      8F          8*32 = 256 bits
*
*
ICMP6_DST_UNREACH      EQU      1          @DBA
ICMP6_PACKET_TOO_BIG   EQU      2          @DBA
ICMP6_TIME_EXCEEDED    EQU      3          @DBA
ICMP6_PARAM_PROB       EQU      4          @DBA
ICMP6_INFOMSG_MASK     EQU      128        @DBA
ICMP6_ECHO_REQUEST     EQU      128        @DBA
ICMP6_ECHO_REPLY       EQU      129        @DBA
MLD_LISTENER_QUERY     EQU      130        @DBA
MLD_LISTENER_REPORT    EQU      131        @DBA
MLD_LISTENER_REDUCTION EQU      132        @DBA
ND_ROUTER_SOLICIT      EQU      133        @DBA
ND_ROUTER_ADVERT       EQU      134        @DBA
ND_NEIGHBOR_SOLICIT    EQU      135        @DBA
ND_NEIGHBOR_ADVERT     EQU      136        @DBA
ND_REDIRECT            EQU      137        @DBA
*
*****
*
*   Routing header                                     @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B418
IP6_RTHDR      DSECT ,
      AGO      .C418
.B418      ANOP      ,
IP6_RTHDR      DS      0F
.C418      ANOP      ,
IP6R_NXT      DS      BL1      Next header
IP6R_LEN      DS      BL1      Length in units of 8 octets
IP6R_TYPE     DS      BL1      Routing type
IP6R_SEGLEFT  DS      BL1      Segments left
*
*****
*
*   Type 0 Routing header                             @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B419
IP6_RTHDR0     DSECT ,
      AGO      .C419
.B419      ANOP      ,
IP6_RTHDR0     DS      0F
.C419      ANOP      ,
IP6R0_NXT     DS      BL1      Next header
IP6R0_LEN     DS      BL1      Length in units of 8 octets
IP6R0_TYPE    DS      BL1      Always zero
IP6R0_SEGLEFT DS      BL1      Segments left
IP6R0_RESERVED DS      FL4      Reserved field
IP6R0_ADDR    DS      0CL16    Upto 127 in6_addr      @DDA
*
*****
*
*   Hop-by-Hop options header                         @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B41A
IP6_HBH      DSECT ,
      AGO      .C41A
.B41A      ANOP      ,
IP6_HBH      DS      0F
.C41A      ANOP      ,
IP6H_NXT     DS      BL1      Next header
IP6H_LEN     DS      BL1      Length in units of 8 octets
IP6H_OPTIONS DS      0C      Options                  @DDA
*
*****
*
*   Destination options header                       @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B41B
IP6_DEST     DSECT ,
      AGO      .C41B

```

```

.B41B      ANOP      ,
IP6_DEST   DS      0F
.C41B      ANOP      , IP6D_NXT      DS      BL1      Next header
IP6D_LEN    DS      BL1      Length in units of 8 octets
IP6D_OPTIONS DS      0C      Options      @DDA
*
*****
*
*      MTU Information      @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B41C
IP6_MTUINFO DSECT ,
      AGO      .C41C
.B41C      ANOP      ,
IP6_MTUINFO DS      0F
.C41C      ANOP      ,
IP6M_ADDR   DS      CL28      Dst address including zone ID
IP6M_MTU     DS      F      Path MTU in host byte order
*
*****
*
*      IPv6 Options Header      @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B41D
IP6_OPT     DSECT ,      AGO      .C41D
.B41D      ANOP      ,
IP6_OPT     DS      0F
.C41D      ANOP      ,
IP60_TYPE   DS      BL1
IP60_LEN     DS      BL1
*
IP6OPT_TYPE EQU x'C0'      @DCA
IP6OPT_TYPE_SKIP EQU x'00' @DCA
IP6OPT_TYPE_DISCARD EQU x'40' @DCA
IP6OPT_TYPE_FORCEICMP EQU x'80' @DCA
IP6OPT_TYPE_ICMP EQU x'C0' @DCA
IP6OPT_MUTABLE EQU x'20' @DCA
*
IP6OPT_PAD1 EQU x'00' @DCA
IP6OPT_PADN EQU x'01' @DCA
*
IP6OPT_JUMBO EQU x'C2' @DCA
IP6OPT_NSAP_ADDR EQU x'C3' @DCA
IP6OPT_TUNNEL_LIMIT EQU x'04' @DCA
IP6OPT_ROUTER_ALERT EQU x'05' @DCA
*
*****
*
*      Jumbo Payload Option      @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B41E
IP6_OPT_JUMBO DSECT ,
      AGO      .C41E
.B41E      ANOP      ,
IP6_OPT_JUMBO DS      0F
.C41E      ANOP      ,
IP60J_TYPE   DS      BL1
IP60J_LEN     DS      BL1
IP60J_JUMBO_LEN DS      4BL1 *
IP6OPT_JUMBO_LEN EQU 6
*
*****
*
*      NSAP Address Option      @DCA *
*
*****
*
      AIF      ('&DSECT' EQ 'NO').B41F
IP6_OPT_NSAP DSECT ,
      AGO      .C41F
.B41F      ANOP      ,
IP6_OPT_NSAP DS      0F
.C41F      ANOP      ,
IP60N_TYPE   DS      BL1
IP60N_LEN     DS      BL1
IP60N_SRC_NSAP_LEN DS      BL1

```

```

IP6ON_DST_NSAP_LEN    DS    BL1
IP6ON_SRC_NSAP        DS    0C
IP6ON_DST_NSAP        DS    0C
*
*****
*
*   Tunnel Limit Option                                @DCA *
*
*****
*
*       AIF    ('&DSECT' EQ 'NO').B420
IP6_OPT_TUNNEL        DSECT ,
*       AGO    .C420
*       .B420  ANOP ,
IP6_OPT_TUNNEL        DS    0F
*       .C420  ANOP ,
IP6OT_TYPE            DS    BL1
IP6OT_LEN             DS    BL1
IP6OT_ENCAP_LIMIT     DS    BL1
*
*****
*
*   Router alert values (in network byte order)        @DCA *
*
*****
*
IP6_ALERT_MLD          EQU    0
IP6_ALERT_RSVP         EQU    1
IP6_ALERT_AN           EQU    2
*
*****
*
*   Source address selection preferences                @DJA
*
*   Used with setsockopt/getsockopt(Socket#IPV6_ADDR_PREFERENCES)
*   and BPX1PCT(PC#IsSrcAddr) for inet6_is_srcaddr() function
*
*****
*
IPV6_PREFER_SRC_HOME   EQU x'00000001'   Prefer home address
IPV6_PREFER_SRC_COA    EQU x'00000002'   Prefer care of address
IPV6_PREFER_SRC_TMP    EQU x'00000004'   Prefer temporary address
IPV6_PREFER_SRC_PUBLIC EQU x'00000008'   Prefer public address
IPV6_PREFER_SRC_CGA    EQU x'00000010'   Prefer Cryptographically
*                                           generated address
IPV6_PREFER_SRC_NONCGA EQU x'00000020'   Prefer non-cryptographically
*                                           generated address
*
*****
*
*   BPX1PCT(PC#IsSrcAddr) Argument for inet6_is_srcaddr() @DJA
*
*   inet6_is_srcaddr(IsSrcAddrIpAddr, IsSrcAddrFlags)
*   is implemented with BPX1PCT(' ',PC#IsSrcAddr,ISSRCADDR#LEN,
*   ISSRCADDR, Rv, Rc, Rsn)
*   where the ISSRCADDR argument is defined as follows:
*
*****
*       AIF    ('&DSECT' EQ 'NO').B426
ISSRCADDR              DSECT ,
*       AGO    .C426
*       .B426  ANOP ,
ISSRCADDR              DS    0F
*       .C426  ANOP ,
ISSRCADDRVER           DS XL1             Version. 1
ISSRCADDRVER1          EQU 1             Version value
*       DS XL3             Reserved. Must be 0
ISSRCADDRIPADDR        DS CL(SOCK#LEN+SOCK_SIN6#LEN) sockaddr_in6
ISSRCADDRFLAGS          DS F             Flags. See IPV6_PREFER_SRC_*
*       DS 6F             Reserved. Must be 0
*
ISSRCADDR#LEN          EQU *-ISSRCADDR Length of ISSRCADDR area
*
** BPXYSOCK End

```

## BPXYSSET – Map the sigaction set

DSECT=.. is not supported. The generated code will allocate SSETOPTION\_FLAGS and a DSECT for SSET. This should be followed by CSECT statement to return to the current DSECT or CSECT.

AMODE 64 callers use “BPXYSSSTF — Map the sigaction set” on page 1211.

```

BPXYSSSTF
** BPXYSSSTF: Macro which enables multiple signal calls
** Used By: SA2
SSETOPTION_FLAGS DS 0F
SSETOPTION_FLAGS1 DS FL1 FLAGS INDICATING CALLER OPTIONS
SSET_IGINVALID EQU X'80' IGNORE INVALID SIGNALS & SIGACTIONS X
                      0=DO NOT IGNORE, 1=IGNORE
                      DS 3FL1 RESERVED
SSET DSECT
SSETCONSOLMASK DS XL8 SIGNALS HAVING THE SAME FLAGS,MASK, X
                      USERDATA, AND SIGNAL ACTION
SSETCOMPARE DS 0CL20
SSETFLAGS DS XL4 VALUE FOR SIGACTION FLAGS (BPXYSIGN)
SSETSAHANDLER DS A ADDRESS OF A SIGNAL HANDLER ROUTINE
SSETSAMASK DS XL8 VALUE FOR SIGACTION MASK
SSETUSERDATA DS F USER DEFINED DATA
SSET#LENGTH EQU *-SSET LENGTH OF ONE SSET ENTRY
** BPXYSSSTF End

```

## BPXYSSSTF — Map response structure for file system status

```

BPXYSSSTF
** BPXYSSSTF: file system status response structure
** Used By: STF STV FTV VSF
SSTF DSECT
SSTFID DC C'SSTF' EBCDIC ID - SSTF (f_0Ecblen)
SSTFLEN DC A(SSTF#LENGTH) Length of SSTF (f_0Ecblen)
SSTFBLOCKSIZE DS F Block size (f_bsize)
                      DS F Reserved
SSTFDBLTOTSPACE DS 0D Name of dblword field - total
                      DS F Reserved
SSTFTOTALSPACE DS F Total space. The total number of X
                      blocks on file system in units of X
                      f_frsize (f_blocks)
SSTFDBLUEDSPACE DS 0D Name of dblword field - used
                      DS F Reserved
SSTFUSEDSPACE DS F Allocated space in block size units X
                      (f_0Eusedspace)
SSTFDBLFREESPACE DS 0D Name of dblword field - free
                      DS F Reserved
SSTFFREESPACE DS F Space available to unprivileged X
                      users in block size units X
                      (f_bavail)
SSTFENDOVER1 EQU * End of Version 1 SSTF
SSTFFSID DS F File system ID (f_fsid) X
                      Set by LFS
SSTFFLAG DS 0BL.32 Bit mask of f_flag vals
SSTFFLAGB1 DS XL1 byte 1
SSTFEXPORTED EQU X'40' Filesys is exported X
                      (ST_0EEXPORTED) X
                      Set by LFS
SSTFV3PROP DS XL1 NFS V3 Properties
SSTFFSF_V3RET EQU X'80' V3 Prop Returned
SSTFFSF_CANSETTIME EQU X'10' time_delta accuracy
SSTFFSF_HOMOGENEOUS EQU X'08' Pathconf same for all
SSTFFSF_SYMLINK EQU X'02' Supports Symlinks
SSTFFSF_LINK EQU X'01' Supports Hard Links
SSTFFLAGB3 DS XL1 byte 3
SSTFFLAGB4 DS XL1 byte 4
SSTFN0SEC EQU X'04' No Security checks enforced
SSTFN0SUID EQU X'02' SetUID/SetGID not supported X
                      (ST_NOSUID) X
                      Set by LFS
SSTFRDONLY EQU X'01' Filesys is read only X
                      (ST_RDONLY) X
                      Set by LFS
SSTFMAXFILESIZE DS 0D Name of dblword field - maximum X
                      file size X
                      May be set by LFS
SSTFMAXFILESIZEHW DS F High word of max file size X
                      (f_0Emaxfilesizehw)
SSTFMAXFILESIZELW DS F Low word of max file size X
                      (f_0Emaxfilesizelw)

```

SSTFENDLFSINFO	DS	CL16	Reserved	
SSTFFRSIZE	EQU	*	End of LFS information	
	DS	F	Fundamental filesystem block size (f_frsize)	X
	DS	F	Reserved	
SSTFDBLBFREE	DS	0D	Name of dblword field - total number of free blocks	X
	DS	F	Reserved	
SSTFBFREE	DS	F	Total number of free blocks (f_bfree)	X
SSTFFILENODES	DS	0CL12	File nodes	
SSTFFILES	DS	F	Total number of file nodes in the file system (f_files)	X
SSTFFFREE	DS	F	Total number of free file nodes (f_ffree)	X
SSTFFAVAIL	DS	F	Number of free file nodes available to unprivileged users (f_favail)	X
SSTFFNAMEMAX	DS	F	Maximum file name len (f_namemax)	
SSTFINVARSEC	DS	F	Number of seconds file system will remain unchanged (f_0Einvarsec)	X
				X
SSTFTIME_DELTA	DS	0CL8	Set file time granularity	
SSTFTIME_DELTA_SEC	DS	F	Seconds	
SSTFTIME_DELTA_NS	DS	F	Nano-seconds	
	DS	CL12	Reserved	
SSTF#LENGTH	EQU	*	SSTF Length of this structure	
SSTF#MINLEN	EQU		SSTFENDVER1-SSTF	
SSTF#LFSLEN	EQU		SSTFENDLFSINFO-SSTF	
** BPXYSTAT End				

## BPXYSTAT — Map the response structure for stat

```

BPXYSTAT ,
** BPXYSTAT: stat system call structure
** Used By: FST LST STA
STAT      DSECT ,
ST_BEGIN  DS      0D
*
ST_EYE    DC      C'STAT' Eye catcher
ST_LENGTH DC      AL2(STAT#LENGTH)
          Length of this structure
ST_VERSION DC     AL2(ST#VER)
          Version of this structure
ST_MODE   DS      F      File Mode, mapped by BPXYMODE
ST_INO    DS      F      File Serial Number
ST_DEV    DS      F      Device ID of the file
ST_NLINK  DS      F      Number of links
ST_UID    DS      F      User ID of the owner of the file
ST_GID    DS      F      Group ID of the Group of the file
ST_SIZE   DS      0D     File Size in bytes, for regular
*          files. Unspecified, for others
ST_SIZE_H DS      F      First word of size
ST_SIZE_L DS      F      Second word of size
ST_ETIME  DS      F      Time of last access
ST_MTIME  DS      F      Time of last data modification
ST_CTIME  DS      F      Time of last file status change
*          Time is in seconds since
*          00:00:00 GMT, Jan. 1, 1970
ST_RDEV   DS      0F     Device Information
ST_MAJORNUMBER DS     H      Major number for this file, if it
*          is a character special file.
ST_MINORNUMBER DS     H      Minor number for this file, if it
*          is a character special file.
ST_AUDITORAUDIT DS     F      Area for auditor audit info
ST_USERAUDIT DS      F      Area for user audit info
ST_BLKSIZE DS      F      File Block size
ST_CREATETIME DS      F      File Creation Time
ST_AUDITID DS      4F     RACF File ID for auditing
ST_RES01   DS      F
ST_CHARSETID DS     0XL12 Coded Character Set ID (obsolete)
ST_FILETAG DS      0F     File Tag
FT_CCSD   DS      H      Coded character set ID in binary
FT_UNTAGGED EQU     X'0000' File has no tag
FT_BINARYTAG EQU     X'FFFF' File is binary data
FT_FLAGS   DS      XL2    File tagging flags
FT_TXTFLAG EQU     X'8000' File is pure text data
FT_DEFERTAG EQU     X'4000' File to be tagged at 1st write

```



```

ST_RES01A      DS      2F      reserved
ST_BLOCKS_D    DS      0D      Double word number - blocks allocated
ST_RES02       DS      F
ST_BLOCKS      DS      F      Number of blocks allocated
ST_GENVALUE    DS      0XL4    General attribute values
ST_OPAQUE      DS      XL3     Opaque attribute flags- Reserved
ST_VISIBLE     DS      X       Visible attribute flags
ST_SHARELIB    EQU     X'10'    Shared Library Flag
ST_NOSHAREAS   EQU     X'08'    No shareas flag
ST_APFAUTH     EQU     X'04'    APF authorized flag
ST_PROGCTL     EQU     X'02'    Program controlled flag
ST_EXTLINK     EQU     X'01'    External Symlink
ST_REFTIME     DS      F       Reference time
ST_FID         DS      2F      File identifier
ST_FILEFMT     DS      XL1     File Format
ST_FSPFLAG2    DS      XL1     IFSP_FLAG2 ACL support
ST_ACCESSACL   EQU     X'80'    Access Acl exists
ST_FMODELACL   EQU     X'40'    File Model Acl exists
ST_DMODELACL   EQU     X'20'    Directory Model Acl exists
ST_RES03       DS      CL2     reserved
ST_CTIMESEC    DS      F       Micro-Sec of full Ctime
ST_SECLABEL    DS      CL8     Security Label
ST_MPTDEV      DS      F       Devno of parent mount point
ST_ENDVER1     EQU     *       End of Ver 1 Stat
*
*
ST_ATIME64     DS      D       Access Time
ST_MTIME64     DS      D       Data Modification Time
ST_CTIME64     DS      D       Meta-data Change Time
ST_CREATETIME64 DS      D       File Creation Time
ST_REFTIME64   DS      D       Reference Time
*
ST_RES05       DS      CL16    Reserved
ST_ENDVER2     EQU     *       End of Ver 2 Stat
*
*   Constants
*
STAT#LENGTH    EQU     *-STAT   Length of STAT
ST#VER01       EQU     1        Version 1 of this structure
ST#VER02       EQU     2        Version 1 of this structure
ST#VER         EQU     ST#VER02 Current version
ST#LEN         EQU     STAT#LENGTH Length of STAT
ST#VER01LEN    EQU     ST_ENDVER1-ST_BEGIN
ST#VER02LEN    EQU     ST_ENDVER2-ST_BEGIN
** BPXYSTAT End

```

## BPXYTCCP – Map the terminal control code page structure

```

                BPXYTCCP
** BPXYTCCP: terminal control code page structure
** Used By: TGC TSC TST
TCCP            DSECT
TCCPFLAG        DS      0BL.32 Bit mask of __tccp_flags
TCCPFLAGB1     DS      XL1     byte 1
TCCPFLAGB2     DS      XL1     byte 2
TCCPFLAGB3     DS      XL1     byte 3
TCCPFLAGB4     DS      XL1     byte 4
TCCPFASTP      EQU     X'02'   If set, indicates that the application can optionally
                                use iconv() services to build the translation tables
                                once and perform all subsequent translation locally.
                                (_TCCP_FASTP)
TCCPBINARY     EQU     X'01'   If set, indicates that binary mode is desired.
                                The code pages are ignored.
                                (_TCCP_BINARY)
TCCPSRCNAME     DS      CL32    Source code page name
                                The code page name is case sensitive and must be null
                                (X'00') terminated.
TCCPTRGNAME     DS      CL32    Target code page name
                                The code page name is case sensitive and must be null
                                (X'00') terminated.
TCCPEND        EQU     *       End of TCCP

```

```

*
*   Constants
*
TCCP#LENGTH      EQU   *-TCCP Length of this structure
*
*   CPCN capability constants
*
TCCP#CPNAMESONLY EQU   1      Code page names only (_CPCN_NAMES)
TCCP#CPNAMESANDBLS EQU   2      Code page names and conversion tables X
                                   (_CPCN_TABLES)
TCCP#CPNAMEMAX    EQU  32      Maximum length of code page name      X
                                   including terminating null          X
                                   (_TCCP_CPNAMEMAX)
** BPXYTCCP End

```

## BPXYTHDQ — Mapping of THDQ structure for BPX1PQG

BPXYTHDQ maps the THDQ structure that is supplied to the BPX1PQG callable service.

```

THDQ      BPXYTHDQ , THDQ - THDQ structure for BPX1PQG callable X
          DSECT      service
THDQHDR   DS      1CL0048 +0 Header section
          ORG      THDQHDR
THDQEYE   DS      1CL0004 +0 eye catcher - 'THDQ'
THDQLENGTH DS 1FL2      +4 Length of THDQ structure
THDQVERSION DS 1FL2      +6 Version number
THDQNUMENTS DS 1FL4      +8 Number of entries in thread array      X
                                   ThdqArray
THDQFLAGS DS 1FL4      +C Flags relating to contents of structure
          ORG      THDQFLAGS
THDQFLAGS1 DS 1FL1      +C 1st flag byte
          ORG      THDQFLAGS1
THDQALLSAFE EQU X'80' All threads are frozen in a safe state
          ORG      THDQFLAGS1+X'00000001'
THDQFLAGS2 DS 1FL1      +D 2nd flag byte
THDQFLAGS3 DS 1FL1      +E 3rd flag byte (used by exit). Cleared on X
                                   initial call to LE exit
THDQFLAGS4 DS 1FL1      +F 4th flag byte
          ORG      THDQFLAGS4
THDQGETSTATE EQU X'80' Get State Data requested by caller (input to X
                                   exit)
          ORG      THDQFLAGS+X'00000004'
THDQEXITWKA DS 1CL0016 +10 Reserved for registered LE exit
          DS      1CL0016 +20 Reserved
THDQDYN    DS      0C      +30 Dynamic section
*
* *****
*
*   * Declare array of thread areas
*
* *****
*
THDQARRAY DS 1CL0256 Array of Thread Areas
          ORG      THDQARRAY
THDQATHID DS 1CL0008 Thread ID of target thread
THDQAFLAGS DS 1FL4      Flags returned for target thread
          ORG      THDQAFLAGS
THDQAFLAGS1 DS 1FL1      Flag1 returned for target thread
          ORG      THDQAFLAGS1
THDQANOTFOUND EQU X'80' Thread was not found, no data was returned
THDQAQFRZSAFE EQU X'40' Thread is now frozen in a safe state      X
                                   determined by Language Env Exit
THDQAOTHERLE EQU X'20' Thread is part of other language environment X
                                   process
THDQANODATA EQU X'10' Status data is not available for this thread X
                                   (if Get State is requested). The PSW/Regs and X
                                   other status info are not valid. The thread X
                                   may be in the process of being created.
THDQACONDWAIT EQU X'08' Task is in Condition Wait.      X
                                   If this bit is set, only the DSA ptr --
                                   Reg13 or Reg4 -- is returned. ThdQAPswIA X
                                   and all the other regs in ThdQAREgs are set X
                                   to 0. (Caller should already have properly X
                                   saved the PSW and registers information.)
          ORG      THDQAFLAGS1+X'00000001'

```

```

THDQAFLAGS2 DS 1FL1      Flag2 returned for target thread
THDQAFLAGS3 DS 1FL1      Flag3 returned for target thread
THDQAFLAGS4 DS 1FL1      Flag4 -- internal flags used by system
      ORG THDQAFLAGS4
THDQAQUICKFRZ EQU X'80'   Thread was frozen without going through the      X
                        RTL SIR or the slow freeze exit routine.          X
                        1=quick freeze (Status stopped) 0=traditional X
                        freeze
THDQAREGSOK EQU X'40'     Registers and PSW retrieved OK
THDQASKIP EQU X'20'       1 = LE exit should not look at this thread.
      ORG THDQAFLAGS+X'00000004'
THDQAREGSSRC DS 1FL2      (Internal) Source from which the regs are      X
                        retrieved. Valid if ThdqARegsOK set.
      DS 1CL0002          Reserved
THDQAREGS DS 1CL0128      Registers of target thread
      ORG THDQAREGS
THDQAREGSH DS 1CL0064     High registers 0-15
THDQAREGSL DS 1CL0064     Low registers 0-15
THDQADOWNSTACKPTR DS 1CL0008 Down Stack start address of target thread X
                        (For XPLINK)
      ORG THDQADOWNSTACKPTR
THDQADSTACKPTRH DS 1AL4   High part of Stack address
THDQADSTACKPTRL DS 1AL4   Low part of Stack address
THDQAUPSTACKPTR DS 1CL0008 Up Stack start address of target thread
      ORG THDQAUPSTACKPTR
THDQAUSTACKPTRH DS 1AL4   High part of Stack address
THDQAUSTACKPTRL DS 1AL4   Low part of Stack address
THDQAPSWIA DS 1CL0008     Instruction address of target thread
      ORG THDQAPSWIA
THDQAPSWIAH DS 1AL4       High part of instruction addr
THDQAPSWIAL DS 1AL4       Low part of instruction addr
THDQACAAPTR DS 1CL0008     LE CAA pointer for thread
      ORG THDQACAAPTR
THDQACAAPTRH DS 1AL4      High part of CAA pointer
THDQACAAPTRL DS 1AL4      Low part of CAA pointer
      DS 1CL0004          Reserved
THDQATCBPTR DS 1AL4       TCB Pointer for target thread
THDQAEXITWKA DS 1CL0008    Thread work area used by LE exit (for PD)
      DS 1CL0064          Reserved
*
*   Declare constants
*
*
THDQ#REGSPPSD EQU 1        Regs from PPSD
THDQ#REGSIRB EQU 2         Regs from IRB
THDQ#REGSUSTA EQU 3        Regs from USTA
THDQ#REGSLS EQU 4          Regs from link stack
THDQ#REGSTCB EQU 5         Regs from TCB/STCB
THDQ#REGSRB EQU 6          Regs from RB/XSB
THDQ#REGSCW EQU 7          Regs for CondWait. Status returned as zeroes
THDQ#ID EQU C'THDQ'        Eye catcher
THDQ#VER EQU 1             Current version of control block
THDQ#VER01 EQU 1           Version 1 of control block
THDQ_LEN EQU *-THDQ

```

## BPXYTHLI — Thread-level information

```

      BPXYTHLI ,
THLI DSECT ,
THLIID DS CL4            EBCDIC ID - Thli
THLISP DS X              Subpool number of this Thli
THLILEN DS FL3           Length of this Thli
THLIFLAGS DS 0BL4       Flag bits
THLIFLAGB1 DS 0B
THLISIGPENDING EQU X'80' Signal pending flag
THLICANCELDISABLED EQU X'40' Cancel request type 0=enabled, 1=disabled
THLICANCELPENDING EQU X'20' Cancel pending for thread
THLICANCELASYNC EQU X'10' Cancellation request state 0 = controlled, 1 X
                        = aysnc
THLIITERATESIR EQU X'08' Use back door signal dlv 0 = Sir can exit 1 = X
                        New sig in PPSD(Iterate Sir)
THLINOSIG EQU X'04'      Suppress signal generation for this socket      X
                        call.
THLITIMEOUTSET EQU X'02' Kernel Time Out Service requested
THLITIMERPOPPED EQU X'01' Kernel Time Out Service timer popped
      ORG THLIFLAGB1+1
THLIFLAGB2 DS 0B
THLIPTQTIMEOUT EQU X'80' If on, invokers of the BPX1PTQ call will      X

```

```

recieve EAGAIN/JRTimeOut if quiece times oyt X
and all threads are not quieced.
THLIFREEZESTOP EQU X'40' Thread has been frozen via Status Stop
THLIDEFERSIGNALS EQU X'20' Defer signals for user
THLIPOSTANDDEFER EQU X'10' Post regardless of key defer delivery until X
key ok
THLITCBEXITPERC EQU X'08' Set by application to allow abends in X
tcbexits to perc to Tcb
THLIIRBNORETRY EQU X'04' Set by NSSIR to indicate the calling IRB is X
not to retry any abends
THLISIGIRBABEND EQU X'02' Abend on sigkill regardless of state
*
* *****
* *
* * Use WorkPtr64 instead of WorkPtr when ThliUseWorkPtr64 is *
* * ON. Thread was pthread_created in 64 bit mode *
* *
* *****
*
*
THLIUSEWORKPTR64 EQU X'01'
ORG THLIFLAGB2+1
THLIFLAGB3 DS 0B
THLIFORKACCTG EQU X'80' Child accounting data based on setuid X
identity
THLIPROPAUTH EQU X'40' Propagate JSCBAUTH to child on fork
THLIUNDUBCALLERONLY EQU X'20' mvspoclp should cleanup caller's *
process only
ORG
THLIFLAGS+4
THLIIPPSD DS F Address of Ppsd
THLISIGMASK DS BL8 Signal mask. Primarliy set by sigprocmask().
THLIPRLI DS A -> Prli. Process related information
*
* *****
* *
* * Use ThliWorkPtr64 if this thread was created via a pthread *
* * create done in amode 64 *
* *
* *****
*
*
THLIWORKPTR DS A -> To user work area specified on X
pthread_create
THLICOMECB DS 0F User communication ECB
THLICOMECBWAIT EQU X'80' ECB wait bit
THLICOMECBPOST EQU X'40' ECB post bit
ORG THLICOMECB+4
THLICOMFLAGS DS 0BL4 ECB control flags
THLICOMFLAGSB1 DS B reserved
THLICOMFLAGSB2 DS B reserved
THLICOMFLAGSB3 DS 0B reserved for user
THLICOMFLAGSU0 EQU X'80' reserved for user
THLICOMFLAGSU1 EQU X'40' reserved for user
THLICOMFLAGSU2 EQU X'20' reserved for user
THLICOMFLAGSU3 EQU X'10' reserved for user
THLICOMFLAGSU4 EQU X'08' reserved for user
THLICOMFLAGSU5 EQU X'04' reserved for user
THLICOMFLAGSU6 EQU X'02' reserved for user
THLICOMFLAGSU7 EQU X'01' reserved for user
ORG THLICOMFLAGSB3+1
THLICOMFLAGSB4 DS 0B
THLIWILLFREEZEME EQU X'08' LE will issue FreezeMe for this task
THLIWFROZEN EQU X'04' BPX1PQG freeze request has been issued X
against the task
THLISIGPOSTED EQU X'02' User posted due to signal
THLISIGWAIT EQU X'01' User wants ECB posted when a signal will be X
delivered
ORG THLICOMFLAGS+4
THLIKEY DS CL1 PSW key of Thli control block. The key is in X
bits 0-3, bits 4-7 are zero
THLIIP@LEN DS X
THLIIFLAGS2 DS 0B Flag Byte
*
* *****
* *
* * Specifying JOBNAME for APPLID is only used by MRPWD and *
* * PRSUI. *
* *
* *****
*

```

```

THLIF2_SETAPPL EQU X'80' Set RACROUTE APPL parm with JOBNAME
THLICVTON EQU X'40' Activates auto conversion for this thread
THLICVTOFF EQU X'20' Deactivates auto conversion for this thread
                                thread-both ThliCvton and ThliCvtOff should
                                not be on
        ORG    THLIFLAGS2+1
THLIECECPARMNUM DS X      Number of parms being passed in exec style
                                parm list. Zero indicates either mvs style
                                parm list or not exec
THLITIMERECEB DS 0F      ECB posted when timer pops from BPX1STE call
THLITIMERECEBWAIT EQU X'80' ECB wait bit
THLITIMERECEBPOST EQU X'40' ECB post bit
        DS     BL.030
        ORG    THLITIMERECEB+4
THLIASPIRBECEB DS 0F      ECB posted when aio done
THLIASPIRBWAIT EQU X'80' Wait bit
THLIASPIRBPOST EQU X'40' Post bit
        ORG    THLIASPIRBECEB+4
THLIJAVA DS A            JAVA thread control block address. *** DO NOT
                                MOVE this field *** Modified by assembler
                                code in JAVA
THLIIP
THLITIMEOUT DS 0CL8       Kernel Time Out Service parameters
THLISECS DS BL4           Seconds to wait for event
THLINANOS DS BL4         Nanoseconds to wait for event
THLIICSID DS H           Program character set Id for filesystem
                                reads/writes
THLIAPLLIDLEN DS X        Len of string in ThliAppid ignored if 0 or >8
        DS     CL5        Reserved
THLISRMFLGS DS 0B         SRM flags Ownership: SRM
THLIVCMOVERRIDE EQU X'80' This bit indicates that this unit of work
                                should not follow the standard SRM management
                                in an VCM=on environment. Instead of trying
                                to assign the work to the same affinity node
                                for cache efficiency concerns, assign this
                                work to any affinity node, ignore any cache
                                concerns.
        ORG    THLISRMFLGS+1
THLISHMDUMPPRIO DS X      Dump priority assigned to shared memory
                                segment on next shmget() or shmat() call
        DS     CL3        Reserved
THLIWORKPTR64 DS CL8      -> To user work area specified on
                                pthread_create
THLIUTOKENINFO DS 0CL8    Fork Hi Memory token
THLIPARENTTKN DS CL4      parents IARV64 user token
THLICHILDTKN DS CL4       token assigned to child
THLIRETCODE DS F          Failing retcode from CPR
THLIRSNCODE DS F          Failing rsnocode from CPR
THLIAPLPID DS CL8         Applid passed for ptsec, and __passwd after
                                ptsec
        DS     CL24       reserved
THLIEP_FUNCTIONCODE DS F
THLIXTENEDPARMS DS 0CL24
        ORG    THLIXTENEDPARMS
THLIXTENEDPARMAREA DS 0CL24
        DS     CL24
*
*
* Add a substructure of 5 under the Main UNION for each
* set of input parms (each of these maps over the storage
* defined as ThliExtendedParmArea)
*
*
*
*
        ORG    THLIXTENEDPARMS
THLIXTENEDAPPLPARMS DS 0CL24 Entry parms
THLIEP_APPLIDLEN DS X
THLIEP_APPLID DS CL8
        DS     CL15
        ORG    THLIXTENEDPARMS
THLIXTENEDMRSMFPARMS DS 0CL24 BPX1SMF parms
THLIEP_MRSMFFLAGS DS 0B
THLIEP_MRSMFIEFU83 EQU X'80' If on for invokers of BPX1SMF, SMF is
                                invoked to write the SMF record such that the
                                SMF record is given to installation exit
                                IEFU83 rather than exit IEFU84
        ORG    THLIEP_MRSMFFLAGS+1
        DS     CL23
        ORG    THLIXTENEDPARMS
THLIXTENEDGIDNAME DS 0CL24 Output for GIDNameSet
THLIEP_GIDNAMELEN DS F

```

```

THLIEP_GIDNAME DS CL8
THLIEP_GIDLEN DS F
THLIEP_GID DS F
THLIEP_GROUPCOUNT DS F
*
*   End of Extended parameter area declarations
*
*
* *****
* * The ThliSecErrDetail area provides detailed information for *
* * select SAF service errors. This information is only provided when*
* * the syscall rv/rc/rs values are ambiguous and do not provide the *
* * user with enough information to determine the potential cause *
* * of the error. Refer to the appropriate SAF service documentation *
* * to decode the RACF/SAF return codes. *
* *****
*
*
THLISECERRDETAIL DS 0CL64
THLISECERRCT DS 0CL40      This level also maps the first 40 bytes of
                           the ctrace SAF exception record, see the
                           bpxtrace command for details
THLISECSERVICE NAME DS CL8 SAF service name
THLISECSERVICEQUAL DS CL8 REQ or function (when applicable)
THLISECSAFRC DS F         SAF return code
THLISECRACFRC DS F         RACF return code
THLISECRACFRS DS F         RACF reason code
THLISECSRV DS F           Syscall RV
THLISESCSRC DS F           Syscall RC
THLISESCRS DS F           Syscall RS
THLISECERRNONCT DS F
THLISECSYSCALL DS CL8      Syscall name
                           DS CL4      reserved (dword bdy)
THLISECERRTOD DS CL8       Time of error (TOD)
                           DS CL20      reserved
*
*
*   NOTE: The size of this control block is retrieved dynamically
*   during runtime by the modules that need it.
*   When adding additional fields to this control block,
*
*   =====> THE ONLY MODULE THAT *MUST* BE RECOMPILED IS BPXPRIT
*
*
THLIEND DS 0C             End of Thli
*
*   Extended parameter codes.
*
*
THLIEP_APPLSET EQU 1
*
*
*
THLI#ID EQU C'THLI'       Control Block Acronym
THLI#LEN EQU 144           Length of Thli
THLI#SP EQU 230            Subpool for the Thli
THLI_LEN EQU *-THLI

```

## BPXYTIMS – Map the response structure for times

```

                BPXYTIMS ,
** BPXYTIMS: times syscall structure
** Used By: TIM
TIMS            DSECT ,
TIMSBEGIN       DS 0F
TIMSUTIME       DS F      User CPU time of current process
*               in hundredths of a second.
*               This includes the TCB and SRB time
*               of the calling process minus the
*               TCB time accumulated while running
*               in the kernel address space.
TIMSSTIME       DS F      System CPU time of current process
*               in hundredths of a second.
*               This is the TCB time accumulated
*               while running in the
*               kernel address space.
TIMSCUTIME       DS F      Sum of user CPU time values (as

```

```

*
* defined in TIMSUTIME) and child user
* CPU time values (as defined in
* TIMSCUTIME) for all waited-for
* child processes. Zero if the
* current process has no waited-for
* children.
TIMSCSTIME      DS      F      Sum of system CPU time values (as
* defined in TIMSSTIME) and child
* system CPU time values (as defined in
* TIMSCSTIME) for all waited-for
* child processes. Zero if the
* current process has no waited-for
* children.
TIMS#LENGTH     EQU     *-TIMS Length of this structure
** BPXYTIOS End

```

## BPXYTIOS – Map the termios structure

Use PREFIX to make the labels unique. The specified characters specified will be appended before each label.

```

                BPXYTIOS      , PREFIX=
** BPXYTIOS: Termios structure
** Used By: TGA TSA TFH TFW
BPXYTIOS DSECT ,                Define DSECT
* baud rate values
B0      EQU      0                0      baud (hang-up)
B50     EQU      1                50     baud
B75     EQU      2                75     baud
B110    EQU      3                110    baud
B134    EQU      4                134.5  baud
B150    EQU      5                150    baud
B200    EQU      6                200    baud
B300    EQU      7                300    baud
B600    EQU      8                600    baud
B1200   EQU      9                1200   baud
B1800   EQU     10                1800   baud
B2400   EQU     11                2400   baud
B4800   EQU     12                4800   baud
B9600   EQU     13                9600   baud
B19200  EQU     14               19200   baud
B38400  EQU     15               38400   baud
* c_cflag offsets for baud rate. These values are
* used to refer to the correct byte within c_cflag. For
* instance, "MVI C_CFLAG+ISPEED_0,B50".
OSPEED_0 EQU      0                Offset to OUTPUT baud rate
ISPEED_0  EQU      1                Offset to INPUT baud rate
* Values for c_cflag field are bitwise distinct except for
* character size bits - which form a number.
CLOCAL    EQU     X'01'            Ignore modem status lines
CREAD     EQU     X'02'            Enable receiver
CSIZE     EQU     X'30'            Character size bits
CS5       EQU     X'00'            B'00' - 5 bits/character
CS6       EQU     X'10'            B'01' - 6 bits/character
CS7       EQU     X'20'            B'10' - 7 bits/character
CS8       EQU     X'30'            B'11' - 8 bits/character
CSTOPB    EQU     X'80'            Send two stop bits, else one
HUPCL     EQU     X'01'            Hang up on last close
PARENB    EQU     X'02'            Parity enable
PARODD    EQU     X'04'            Odd parity, else even
PACKET    EQU     X'08'            Packet mode enabled
PKT3270   EQU     X'10'            3270 Passthru mode allowed
PTU3270   EQU     X'20'            3270 Passthru mode enabled
PKTXTND   EQU     X'40'            Extended Packet mode enabled
* c_cflag offsets for bits previously defined. These values are
* used to refer to the correct byte within c_cflag. For
* instance, "TM C_CFLAG+HUPCL_0,HUPCL".
CLOCAL_0  EQU      3
CREAD_0   EQU      3
CSIZE_0   EQU      3
CS5_0     EQU      3
CS6_0     EQU      3
CS7_0     EQU      3
CS8_0     EQU      3
CSTOPB_0  EQU      3
HUPCL_0   EQU      2

```

```

PARENB_0 EQU 2
PARODD_0 EQU 2
PACKET_0 EQU 2
PKT3270_0 EQU 2
PTU3270_0 EQU 2
PKTXTND_0 EQU 2
* Values for c_lflag field are bitwise distinct.
ECHO EQU X'08' Enable echo
ECHOE EQU X'02' Echo ERASE as error correcting X
backspace
ECHOK EQU X'04' Echo KILL
ECHONL EQU X'01' Echo new line
ICANON EQU X'10' Canonical input
IEXTEN EQU X'20' Enable extended functions
ISIG EQU X'40' Enable signals
NOFLSH EQU X'80' Disable flush after interrupt, X
quit, or suspend
TOSTOP EQU X'40' Send SIGTTOU for background X
output
XCASE EQU X'80' Canonical Upper/Lower X
presentation
* c_lflag offsets for bits previously defined. These values are
* used to refer to the correct byte within c_lflag. For
* instance, "TM C_LFLAG+TOSTOP_0,TOSTOP".
ECHO_0 EQU 3
ECHOE_0 EQU 3
ECHOK_0 EQU 3
ECHONL_0 EQU 3
ICANON_0 EQU 3
IEXTEN_0 EQU 3
ISIG_0 EQU 3
NOFLSH_0 EQU 0
TOSTOP_0 EQU 1
XCASE_0 EQU 3
* Values for c_iflag field are bitwise distinct.
BRKINT EQU X'01' Signal interrupt on break
ICRNL EQU X'02' Map CR to NL on input
IGNBRK EQU X'04' Ignore break condition
IGNCR EQU X'08' Ignore CR
IGNPAR EQU X'10' Ignore characters with parity X
errors
INLCR EQU X'20' Map NL to CR in input
INPCK EQU X'40' Enable input parity check
ISTRIP EQU X'80' Strip character
IXOFF EQU X'01' Enable start/stop input X
control
IXON EQU X'02' Enable start/stop output X
control
PARMRK EQU X'04' Mark parity errors
IUCLC EQU X'08' Map UC->LC on input
IXANY EQU X'10' Any char restarts output
* c_iflag offsets for bits previously defined. These values are
* used to refer to the correct byte within c_iflag. For
* instance, "TM C_IFLAG+BRKINT_0,BRKINT".
BRKINT_0 EQU 3
ICRNL_0 EQU 3
IGNBRK_0 EQU 3
IGNCR_0 EQU 3
IGNPAR_0 EQU 3
INLCR_0 EQU 3
INPCK_0 EQU 3
ISTRIP_0 EQU 3
IXOFF_0 EQU 2
IXON_0 EQU 2
PARMRK_0 EQU 2
IUCLC_0 EQU 2
IXANY_0 EQU 2
* Values for c_oflag are bitwise distinct.
OPOST EQU X'01' Perform output processing
OLCUC EQU X'02' Map LC->UC on output
ONLCR EQU X'04' Map NL->CR on output
OCRNL EQU X'08' Map CR->NL on output
ONOCR EQU X'10' No CR at column 0
ONLRET EQU X'20' NL performs CR function
OFILL EQU X'40' Use fill chars for delay
OFDEL EQU X'80' Use DEL, not NUL, for fill
NLDLY EQU X'01' Newline delay type
NL0 EQU X'00' NL delay type 0
NL1 EQU X'01' NL delay type 1
TABDLY EQU X'0C' Tab delay type
TAB0 EQU X'00' Tab delay type 0
TAB1 EQU X'04' Tab delay type 1

```



```

TAB2      EQU X'08'      Tab delay type 2
TAB3      EQU X'0C'      Expand tabs to spaces
CRDLY     EQU X'30'      CR delay type
CR0       EQU X'00'      CR delay type 0
CR1       EQU X'10'      CR delay type 1
CR2       EQU X'20'      CR delay type 2
CR3       EQU X'30'      CR delay type 3
FFDLY     EQU X'40'      Form-feed delay type
FF0       EQU X'00'      FF delay type 0
FF1       EQU X'40'      FF delay type 1
BSDLY     EQU X'80'      Backspace delay type
BS0       EQU X'00'      BS delay type 0
BS1       EQU X'80'      BS delay type 1
VTDLY     EQU X'01'      Vertical-tab delay type
VT0       EQU X'00'      VT delay type 0
VT1       EQU X'01'      VT delay type 1
*
* c_oflag offsets for bits previously defined. These values are
* used to refer to the correct byte within c_oflag. For
* instance, "TM C_OFLAG+OPOST_0,OPOST".
*
OPOST_0   EQU 3
OLCUC_0   EQU 3
ONLCR_0   EQU 3
OCRNL_0   EQU 3
ONOCR_0   EQU 3
ONLRET_0  EQU 3
OFILL_0   EQU 3
OFDEL_0   EQU 3
NLDLY_0   EQU 2
NL0_0     EQU 2
NL1_0     EQU 2
TABDLY_0  EQU 2
TAB0_0    EQU 2
TAB1_0    EQU 2
TAB2_0    EQU 2
TAB3_0    EQU 2
CRDLY_0   EQU 2
CR0_0     EQU 2
CR1_0     EQU 2
CR2_0     EQU 2
CR3_0     EQU 2
FFDLY_0   EQU 2
FF0_0     EQU 2
FF1_0     EQU 2
BSDLY_0   EQU 2
BS0_0     EQU 2
BS1_0     EQU 2
VTDLY_0   EQU 1
VT0_0     EQU 1
VT1_0     EQU 1
* Optional actions used by tcsetattr()
TCSANOW   EQU 0      Change occurs immediately
TCSADRAIN EQU 1      Change occurs after all output      X
                    has been written
TCSAFLUSH EQU 2      Change occurs after all output      X
                    has been written and input          X
                    has been discarded
* queue selector values for tcflush
TCIFLUSH  EQU 0      Flush data received but not read
TCOFLUSH  EQU 1      Flush data written but not sent
TCIOFLUSH EQU 2      Flush both data received but not    X
                    read and data written but not sent
* action values for tcflow()
TCOOFF    EQU 0      Suspend output
TCOON     EQU 1      Restart suspended output
TCIOFF    EQU 2      Transmit STOP character
TCION     EQU 3      Transmit START character
* Special Control Characters subscripts for cc_c
* field
VINTR     EQU 0      INTR character
VQUIT     EQU 1      QUIT character
VERASE    EQU 2      ERASE character
VKILL     EQU 3      KILL character
VEOF      EQU 4      EOF character
VEOL      EQU 5      EOL character
VMIN      EQU 6      MIN value
VSTART    EQU 7      START character
VSTOP     EQU 8      STOP character
VSUSP     EQU 9      SUSP character
VTIME     EQU 10     TIME value

```

## BPXYUTSN

```
NCCS      EQU 11                      Number of special control chars
C_CFLAG   DC F'0'                     Control modes
C_IFLAG    DC F'0'                     Input modes
C_LFLAG    DC F'0'                     Local modes
C_OFLAG    DC F'0'                     Output modes
C_CC       DC (NCCS)X'0'               Control characters and values
BPXYTIOS#LENGTH EQU *-BPXYTIOS Length of this structure
** BPXYTIOS End
```

## BPXYUTSN — Map the response structure for uname

```
                BPXYUTSN
** BPXYUTSN: uname() structure
** Used By: UNA
UTSN
UTSNAMESYSNAMELEN DS F Length of UTSNAMESYSNAME string
UTSNAMESYSNAME DS CL16 Name of this implementation of the
* operating system (MVS)
UTSNAMENODENAMELEN DS F Length of UTSNAMENODENAME string
UTSNAMENODENAME DS CL32 Name of this node within the
* communications network
UTSNAMERELEASELEN DS F Length of UTSNAMERELEASE string
UTSNAMERELEASE DS CL8 Current release level of this
* implementation
UTSNAMEVERSIONLEN DS F Length of UTSNAMEVERSION string
UTSNAMEVERSION DS CL8 Current version level of this release
UTSNAMEMACHINELEN DS F Length of UTSNAMEMACHINE string
UTSNAMEMACHINE DS CL16 Name of the hardware type on which
* the system is running
UTSN#LENGTH EQU *-UTSN Length of this structure
** BPXYUTSN End
```

## BPXYWAST — Map the wait status word

```
                BPXYWAST
** BPXYWAST: Wait status word
** Used By: EXI MPC WAT
WAST
WASTEXITSTATUS DS XL2 Reserved - set to zeros
* DS 0XL2 Exit Status value passed on the
* BPX1EXI or BPX1MPC system calls
WASTEXITCODE DS 0XL1 Exit return code for ending process
WASTSIGSTOP DS XL1 Signal that stopped process
WASTSIGTERM DS 0XL1 Signal that terminated process
WASTSTOPFLAG DS XL1 Special flag value that qualifies the
* reason for the process being stopped
* or if the process is continued
* from stop, the value would be
* set to WastStopFlagContinued
* * WASTSTOPFLAG Values * * * * *
WASTDUMP EQU X'80' Bit 0 of WASTSTOPFLAG on, a core dump
* was taken when the process terminated
WASTSTOPFLAGSIG EQU X'7F' Process stopped for a signal
WASTSTOPFLAGFORK EQU X'7E' Process stopped for a fork
WASTSTOPFLAGEXEC EQU X'7D' Process stopped for an exec
WASTSTOPFLAGLOCALFORK EQU X'7B' Process stopped for a local fork
WASTSTOPFLAGEXTENDED EQU X'7A' Process stopped for extended event
*
WASTSTOPFLAGCONTINUED EQU X'79' Process continued from stop
WASTSTOPFLAGLOAD EQU X'78' Process stopped for a loadhfs
WASTSTOPFLAGDELETE EQU X'77' Process stopped for a deleteHFS
WAST#LENGTH EQU *-WAST Length of this structure
** BPXYWAST End
```

## BPXYWLM — WLM constants and parameter list DSECTs

BPXYWLM work load manager constants and DSECTs. AMODE 64 callers use “BPXYWLM — WLM constants and parameter list DSECTs” on page 1212.

```

        BPXYWLM
** BPXYWLM: BPX1WLM Interface Declares
** Used By: Callers of the BPX1WLM Interface
*
*   BPX1WLM Function Code Constants
*
WLM_QUERY_METRICS      EQU 1
WLM_QUERY_SCHENV      EQU 2
WLM_CHECK_SCHENV      EQU 3
WLM_DISCONNECT        EQU 4
WLM_DELETE_WORKUNIT   EQU 5
WLM_JOIN_WORKUNIT     EQU 6
WLM_LEAVE_WORKUNIT    EQU 7
WLM_CONNECT_WORKMGR   EQU 8
WLM_CONNECT_SERVERMGR EQU 9
WLM_CREATE_WORKUNIT   EQU 10
WLM_CONTINUE_WORKUNIT EQU 11
WLM_EXTRACT_WORKUNIT  EQU 12
WLM_EXPORT_WORKUNIT   EQU 13
WLM_UNDOEXPORT_WORKUNIT EQU 14
WLM_IMPORT_WORKUNIT   EQU 15
WLM_UNDOIMPORT_WORKUNIT EQU 16
WLM_QUERY_ENCLAVECLASS EQU 17
WLM_CONNECT_EXPORTIMPORT EQU 18
* Function codes 100-112 are reserved
ARM_BIND_THREAD       EQU 200
ARM_BLOCK_TRANSACTION EQU 201
ARM_DESTROY_APPLICATION EQU 202
ARM_DISCARD_TRANSACTION EQU 203
ARM_GENERATE_CORRELATOR EQU 204
ARM_GET_ARRIVAL_TIME  EQU 205
ARM_REGISTER_APPLICATION EQU 206
ARM_REGISTER_METRIC   EQU 207
ARM_REGISTER_TRANSACTION EQU 208
ARM_REPORT_TRANSACTION EQU 209
ARM_START_APPLICATION EQU 210
ARM_START_TRANSACTION EQU 211
ARM_STOP_APPLICATION  EQU 212
ARM_STOP_TRANSACTION  EQU 213
ARM_UNBIND_THREAD     EQU 214
ARM_UNBLOCK_TRANSACTION EQU 215
ARM_UPDATE_TRANSACTION EQU 216
EWLM_CLASSIFY_CORRELATOR EQU 217
*   BPX1WLM/BPX4WLM Parameter List Mappings
*
_WQM          DSECT ,      WLM_QUERY_METRICS Parameter List
_WQM_SYSI_PTR DS    A      Address of a fullword pointer that
*                          contains the address of the buffer
*                          to return the WLM system information.
*                          This data is returned in the format
*                          of the IWMWSYSI mapping macro.
_WQM_SYSI_LEN DS    A      Address of a fullword that contains
*                          the length of the buffer to return
*                          the WLM system information
_WQM_END      DS    0C     End of WQM
*
_WQS          DSECT ,      WLM_QUERY_SCHENV Parameter List
_WQS_SETH_PTR DS    A      Address of a fullword pointer that
*                          contains the address of the buffer
*                          to return the WLM scheduling
*                          environment information.
*                          This data is returned in the format
*                          of the IWMSET mapping macro.
_WQS_SETH_LEN DS    A      Address of a fullword that contains
*                          the length of the buffer to return
*                          the WLM scheduling environment data.
_WQS_END      DS    0C     End of _WQS
*
_WCS          DSECT ,      WLM_CHECK_SCHENV Parameter List
_WCS_SCH_ENV DS    A      Address of a 16 byte character string
*                          that contains the scheduling
*                          environment to be checked.
_WCS_SYS_NAME DS    A      Address of a 8 byte character string

```

*			that contains the system name to be checked.
*_WCS_END	DS	0C	End of _WCS
*			
*_WDC	DSECT	,	WLM_DISCONNECT Parameter List
*_WDC_CONN_TKN	DS	A	Address of a fullword that contains the connect token to be disconnected from.
*			
*_WDC_END	DS	0C	End of _WDC
*			
*_WDW	DSECT	,	WLM_DELETE_WORKUNIT Parameter List
*_WDW_ENC_TKN	DS	A	Address of a doubleword that contains the WLM enclave token representing the work unit to be deleted.
*			
*_WDW_END	DS	0C	End of _WDW
*			
*_WJW	DSECT	,	WLM_JOIN_WORKUNIT Parameter List
*_WJW_ENC_TKN	DS	A	Address of a doubleword that contains the WLM enclave token representing the work unit to join.
*			
*_WJW_END	DS	0C	End of _WJW
*			
*_WLW	DSECT	,	WLM_LEAVE_WORKUNIT Parameter List
*_WLW_ENC_TKN	DS	A	Address of a doubleword that contains the WLM enclave token representing the work unit to leave.
*			
*_WLW_END	DS	0C	End of _WLW
*			
*_WNW	DSECT	,	WLM_CONTINUE_WORKUNIT Parameter List
*_WNW_ENC_TKN	DS	A	Address of a doubleword to return the the WLM enclave token of the created work unit.
*			
*_WNW_END	DS	0C	End of _WNW
*			
*_WCW	DSECT	,	WLM_CREATE_WORKUNIT Parameter List
*_WCW_ENC_TKN	DS	A	Address of a doubleword to return the the WLM enclave token of the created work unit.
*			
*_WCW_CLASSIFY	DS	A	Address of a fullword pointer that contains the address of a IWMCLSFY Parameter List.
*			
*_WCW_ARR_TIME	DS	A	Address of a doubleword field that contains the arrival time of the work request in STCK format.
*			
*_WCW_FUNC_NAME	DS	A	Address of a 8 byte character string that contains the descriptive function name of the work request.
*			
*_WCW_END	DS	0C	End of _WCW
*			
*_WSC	DSECT	,	WLM_CONNECT_SERVERMGR Parameter List
*_WSC_SUB_SYS	DS	A	Address of a 4 byte character string that contains the subsystem type the server manager is requesting connection for.
*			
*_WSC_SUB_SYS_NM	DS	A	Address of a 8 byte character string that contains the subsystem name the server manager is requesting connection for.
*			
*_WSC_APPL_ENV	DS	A	Address of a 32 byte character string that contains the application environment name associated with the server.
*			
*_WSC_PAR_EU	DS	A	Address of a fullword that contains number of parallel execution units in the server environment.
*			
*_WSC_END	DS	0C	End of _WSC
*			
*_WWC	DSECT	,	WLM_CONNECT_WORKMGR Parameter List
*_WWC_SUB_SYS	DS	A	Address of a 4 byte character string that contains the subsystem type the work manager is requesting connection for.
*			
*_WWC_SUB_SYS_NM	DS	A	Address of a 8 byte character string that contains the subsystem name the work manager is requesting connection for.
*			
*_WWC_END	DS	0C	End of _WWC
*			
*_WEW	DSECT	,	WLM_EXTRACT_WORKUNIT Parameter List
*_WEW_ENC_TKN	DS	A	Address of a doubleword that contains the WLM enclave token representing the active work unit.
*			
*			

_WEW_END	DS	0C	End of _WEW
* _WXW _WXW_ENC_TKN	DSECT , DS	A	WLM_EXPORT_WORKUNIT Parameter List Address of a doubleword that contains the WLM enclave token representing the work unit to be exported.
* _WXW_EXP_TKN	DS	A	Address of the 32 bytes to return the WLM export token of the exported work unit.
* _WXW_CONN_TKN	DS	A	Address of a fullword that contains the connect token associated with the workmanager.
* _WXW_END	DS	0C	End of _WXW
* _WUXW _WUXW_EXP_TKN	DSECT , DS	A	WLM_UNEXPORT_WORKUNIT Parameter List Address of the 32 bytes that contains the WLM export token representing the exported work unit.
* _WUXW_CONN_TKN	DS	A	Address of a fullword that contains the connect token associated with the workmanager.
* _WUXW_END	DS	0C	End of _WUXW
* _WIW _WIW_EXP_TKN	DSECT , DS	A	WLM_IMPORT_WORKUNIT Parameter List Address of the 32 bytes that contains the WLM export token representing the exported work unit.
* _WIW_ENC_TKN	DS	A	Address of a doubleword to return the WLM enclave token of the imported work unit.
* _WIW_CONN_TKN	DS	A	Address of a fullword that contains the connect token associated with the workmanager.
* _WIW_END	DS	0C	End of _WIW
* _WUIW _WUIW_EXP_TKN	DSECT , DS	A	WLM_UNIMPORT_WORKUNIT Parameter List Address of the 32 bytes that contains the WLM export token representing the imported work unit.
* _WUIW_CONN_TKN	DS	A	Address of a fullword that contains the connect token associated with the workmanager.
* _WUIW_END	DS	0C	End of _WUIW
* _WQEC _WQEC_ENC_TKN	DSECT , DS	A	WLM_QUERY_ENCLAVECLASS Parameter List Address of a doubleword that contains the WLM enclave token representing the work unit to be queried.
* _WQEC_SYSEC_PTR	DS	A	Address of a fullword pointer that contains the address of the buffer to return the WLM Query Enclave Data. This data is returned in the format of the IWMECD mapping macro.
* _WQEC_SYSEC_LEN	DS	A	Address of a fullword that contains the length of the buffer to return the WLM Query Enclave Data.
* _WQEC_END	DS	0C	End of WQEC
* _WCEI _WCEI_SUB_SYS	DSECT , DS	A	WLM_CONNECT_EXPORTIMPORT Parameter List Address of a 4 byte character string that contains the subsystem type the work manager is requesting connection for.
* _WCEI_SUB_SYS_NM	DS	A	Address of a 8 byte character string that contains the subsystem name the work manager is requesting connection for.
* _WCEI_END	DS	0C	End of _WCEI
* _ABI _ABI_CONTEXT	DSECT , DS	A	ARM_BIND_THREAD Parameter List Reserved.
* _ABI_TRAN_HDL	DS	A	Must be zero. Address of a 8 byte field that contains the transaction handle.
* _ABI_FLAGS	DS	A	Address of a 4 byte field that contains flags.
* _ABI_BUFFER4	DS	A	Address of a data area that contains additional input data.
* _ABI_END	DS	0C	End of _ABI
* _ABT	DSECT ,		ARM_BLOCK_TRANSACTION Parameter

* _ABT_CONTEXT	DS	A	List Reserved.
* _ABT_TRAN_HDL	DS	A	Must be zero. Address of a 8 byte field that contains the transaction handle.
* _ABT_FLAGS	DS	A	Address of a 4 byte field that contains flags.
* _ABT_BUFFER4	DS	A	Address of a data area that contains additional input data.
* _ABT_BLOCK_HDL	DS	A	Address of a fullword pointer that contains the address of the 8 byte field to return the block handle.
* _ABT_END	DS	0C	End of _ABT
* _ADA	DSECT	,	ARM_DESTROY_APPLICATION Parameter List
* _ADA_CONTEXT	DS	A	Reserved.
* _ADA_APPL_ID	DS	A	Must be zero. Address of a 16 byte field that contains the application ID.
* _ADA_FLAGS	DS	A	Address of a 4 byte field that contains flags.
* _ADA_BUFFER4	DS	A	Address of a data area that contains additional input data.
* _ADA_END	DS	0C	End of _ADA
* _ADT	DSECT	,	ARM_DISCARD_TRANSACTION Parameter List
* _ADT_CONTEXT	DS	A	Reserved.
* _ADT_TRAN_HDL	DS	A	Must be zero. Address of a 8 byte field that contains the transaction handle.
* _ADT_FLAGS	DS	A	Address of a 4 byte field that contains flags.
* _ADT_BUFFER4	DS	A	Address of a data area that contains additional input data.
* _ADT_END	DS	0C	End of _ADT
* _AGC	DSECT	,	ARM_GENERATE_CORRELATOR Parameter List
* _AGC_CONTEXT	DS	A	Reserved.
* _AGC_APP_HDL	DS	A	Must be zero. Address of a 8 byte field that contains the application handle.
* _AGC_TRAN_ID	DS	A	Address of a 16 byte field that contains the transaction ID.
* _AGC_PAR_CORR	DS	A	Address of a data area that contains the parent correlator.
* _AGC_FLAGS	DS	A	Address of a 4 byte field that contains flags.
* _AGC_BUFFER4	DS	A	Address of a data area that contains additional input data.
* _AGC_CUR_CORR	DS	A	Address of a fullword pointer that contains the address of the buffer to return the current correlator.
* _AGC_END	DS	0C	End of _AGC
* _AGT	DSECT	,	ARM_GET_ARRIVAL_TIME Parameter List
* _AGT_CONTEXT	DS	A	Reserved.
* _AGT_TIMESTAMP	DS	A	Must be zero. Address of a fullword pointer that contains the address of a 64 bit field to return the arrival time.
* _AGT_END	DS	0C	End of _AGT
* _ARA	DSECT	,	ARM_REGISTER_APPLICATION Parameter List
* _ARA_CONTEXT	DS	A	Reserved.
* _ARA_APP_NAME	DS	A	Must be zero. Address of a character string that contains the application name.
* _ARA_IN_APP_ID	DS	A	Address of a 16 byte field that contains an input application ID.
* _ARA_FLAGS	DS	A	Address of a 4 byte field that contains flags.
* _ARA_BUFFER4	DS	A	Address of a data area that contains additional input data.
* _ARA_OUT_APP_ID	DS	A	Address of a fullword pointer that contains the address of a 16 byte field to return the output application ID.
* *			

_ARA_END	DS	0C	End of _ARA
*_AMR	DSECT	,	ARM_REGISTER_METRIC Parameter List
*_AMR_CONTEXT	DS	A	Reserved.
*_AMR_APP_ID	DS	A	Must be zero.
*_AMR_MET_NAME	DS	A	Address of a 16 byte field that contains the application ID.
*_AMR_MET_FORMAT	DS	A	Address of a character string that contains the metric name.
*_AMR_MET_USAGE	DS	A	Address of a 1 byte field that contains the metric format.
*_AMR_UNIT	DS	A	Address of a 2 byte field that contains the metric usage.
*_AMR_IN_MET_ID	DS	A	Address of a character string that contains the units of the metric.
*_AMR_FLAGS	DS	A	Address of a 16 byte field that contains an input metric ID.
*_AMR_BUFFER4	DS	A	Address of a 4 byte field that contains flags.
*_AMR_OUT_MET_ID	DS	A	Address of a data area that contains additional input data.
*_AMR_END	DS	0C	Address of a fullword pointer that contains the address of a 16 byte field to return the output metric ID.
*_ART	DSECT	,	End of _AMR
*_ART_CONTEXT	DS	A	ARM_REGISTER_TRANSACTION Parameter List
*_ART_APP_ID	DS	A	Reserved.
*_ART_TRAN_NAME	DS	A	Must be zero.
*_ART_IN_TRAN_ID	DS	A	Address of a 16 byte field that contains the application ID.
*_ART_FLAGS	DS	A	Address of a character string that contains the transaction name.
*_ART_BUFFER4	DS	A	Address of a 16 byte field that contains an input transaction ID.
*_ART_OUT_TRAN_ID	DS	A	Address of a 4 byte field that contains flags.
*_ART_END	DS	0C	Address of a data area that contains additional input data.
*_ATR	DSECT	,	Address of a fullword pointer that contains the address of a 16 byte field to return the output transaction ID.
*_ATR_CONTEXT	DS	A	End of _ART
*_ATR_APP_HDL	DS	A	ARM_REPORT_TRANSACTION Parameter List
*_ATR_TRAN_ID	DS	A	Reserved.
*_ATR_TRAN_STA	DS	A	Must be zero.
*_ATR_RESP_TIME	DS	A	Address of a 8 byte field that contains the application handle.
*_ATR_STOP_TIME	DS	A	Address of a 16 byte field that contains the transaction ID.
*_ATR_PAR_CORR	DS	A	Address of a 4 byte field that contains the transaction status.
*_ATR_CUR_CORR	DS	A	Address of a 64 bit field that contains the response time.
*_ATR_FLAGS	DS	A	Address of a 64 bit field that contains the stop time.
*_ATR_BUFFER4	DS	A	Address of a data area that contains the parent correlator.
*_ATR_END	DS	0C	Address of a data area that contains the current correlator.
*_AAS	DSECT	,	Address of a 4 byte field that contains flags.
*_AAS_CONTEXT	DS	A	Address of a data area that contains additional input data.
*_AAS_APP_ID	DS	A	End of _ATR
*_AAS_APP_GRP	DS	A	ARM_START_APPLICATION Parameter List
*_AAS_APP_INS	DS	A	Reserved.
			Must be zero.
			Address of a 16 byte field that contains the application ID.
			Address of a character string that contains the application group name.
			Address of a character string that

*			contains the application
*			instance name.
_AAS_FLAGS	DS	A	Address of a 4 byte field
*			that contains flags.
_AAS_BUFFER4	DS	A	Address of a data area that
*			contains additional input data.
_AAS_APP_HDL	DS	A	Address of a fullword pointer that
*			contains the address of the 8 byte
*			field to return the application
*			handle.
_AAS_END	DS	0C	End of _AAS
*			
_AST	DSECT	,	ARM_START_TRANSACTION Parameter
*			List
_AST_CONTEXT	DS	A	Reserved.
*			Must be zero.
_AST_APP_HDL	DS	A	Address of a 8 byte field that
*			contains the application handle.
_AST_TRAN_ID	DS	A	Address of a 16 byte field that
*			contains the transaction ID.
_AST_PAR_CORR	DS	A	Address of a data area that
*			contains the parent correlator.
_AST_FLAGS	DS	A	Address of a 4 byte field
*			that contains flags.
_AST_BUFFER4	DS	A	Address of a data area that
*			contains additional input data.
_AST_TRAN_HDL	DS	A	Address of a fullword pointer that
*			contains the address of the 8 byte
*			field to return the transaction
*			handle.
_AST_CUR_CORR	DS	A	Address of a fullword pointer that
*			contains the address of the buffer
*			to return the current correlator.
_AST_END	DS	0C	End of _AST
*			
_APA	DSECT	,	ARM_STOP_APPLICATION Parameter
*			List
_APA_CONTEXT	DS	A	Reserved.
*			Must be zero.
_APA_APP_HDL	DS	A	Address of a 8 byte field that
*			contains the application handle.
_APA_FLAGS	DS	A	Address of a 4 byte field
*			that contains flags.
_APA_BUFFER4	DS	A	Address of a data area that
*			contains additional input data.
_APA_END	DS	0C	End of _APA
*			
_APT	DSECT	,	ARM_STOP_TRANSACTION Parameter
*			List
_APT_CONTEXT	DS	A	Reserved.
*			Must be zero.
_APT_TRAN_HDL	DS	A	Address of a 8 byte field that
*			contains the transaction handle.
_APT_TRAN_STA	DS	A	Address of a 4 byte number that
*			contains the transaction status.
_APT_FLAGS	DS	A	Address of a 4 byte field
*			that contains flags.
_APT_BUFFER4	DS	A	Address of a data area that
*			contains additional input data.
_APT_END	DS	0C	End of _APT
*			
_AUB	DSECT	,	ARM_UNBIND_THREAD Parameter
*			List
_AUB_CONTEXT	DS	A	Reserved.
*			Must be zero.
_AUB_TRAN_HDL	DS	A	Address of a 8 byte field that
*			contains the transaction handle.
_AUB_FLAGS	DS	A	Address of a 4 byte field
*			that contains flags.
_AUB_BUFFER4	DS	A	Address of a data area that
*			contains additional input data.
_AUB_END	DS	0C	End of _AUB
*			
_AUT	DSECT	,	ARM_UNBLOCK_TRANSACTION Parameter
*			List
_AUT_CONTEXT	DS	A	Reserved.
*			Must be zero.
_AUT_TRAN_HDL	DS	A	Address of a 8 byte field that
*			contains the transaction handle.
_AUT_BLOCK_HDL	DS	A	Address of a 8 byte field
*			that contains the block handle.
_AUT_FLAGS	DS	A	Address of a 4 byte field



```

*
_AUT_BUFFER4      DS    A      that contains flags.
*                  Address of a data area that
_AUT_END          DS    0C      contains additional input data.
*                  End of _AUT
*
_AUP              DSECT ,      ARM_UPDATE_TRANSACTION Parameter
*                  List
_AUP_CONTEXT      DS    A      Reserved.
*                  Must be zero.
_AUP_TRAN_HDL     DS    A      Address of a 8 byte field that
*                  contains the transaction handle.
_AUP_FLAGS        DS    A      Address of a 4 byte field
*                  that contains flags.
_AUP_BUFFER4      DS    A      Address of a data area that
*                  contains additional input data.
_AUP_END          DS    0C      End of _AUP
*
_ACC              DSECT ,      EWLM_CLASSIFY_CORRELATOR Parameter
*                  List
_ACC_CONTEXT      DS    A      Reserved.
*                  Must be zero.
_ACC_APP_HDL      DS    A      Address of a 8 byte field that
*                  contains the application handle.
_ACC_TRAN_ID      DS    A      Address of a 16 byte field that
*                  contains the transaction ID.
_ACC_FLAGS        DS    A      Address of a 4 byte field
*                  that contains flags.
_ACC_BUFFER4      DS    A      Address of a data area that
*                  contains additional input data.
_ACC_CLASS_CORR   DS    A      Address of a fullword pointer that
*                  contains the address of the buffer to
*                  return the classify correlator.
_ACC_END          DS    0C      End of _ACC
** BPXYWLM End

```

## BPXYWNSZ – Map the winsize structure

BPXYWNSZ maps window/terminal size information. It corresponds to the C winsize structure, which is in `sys/ioctl.h`.

```

                BPXYWNSZ ,
** BPXYWNSZ: Winsize structure
** Used By: ioctl with TIOCGWINSZ and TIOCSWINSZ
BPXYWNSZ DSECT ,      Define DSECT
WS_ROW      DC    H'0'      Rows, in characters
WS_COL      DC    H'0'      Columns, in characters
WS_XPIXEL   DC    H'0'      Horizontal size, pixels
WS_YPIXEL   DC    H'0'      Vertical size, pixels
BPXYWNSZ#LENGTH EQU    *-BPXYWNSZ Length of this structure
** BPXYWNSZ End

```

## BPXZOAPB – z/OS UNIX address space per-process extension

BPXZOAPB maps z/OS UNIX space per-process extension. Only the following fields are externally documented. All other fields are reserved for IBM use only.

- OapbDefaultUseridLen
- OapbDefaultUserid
- OapbDefaultGroupidLen
- OapbDefaultGroupid
- OapbUIDs
- OapbRUid
- OapbEUid
- OapbSUid
- OapbGIDs

- OapbRGid
- OapbEGid
- OapbSGid
- OapbContainerIdentifiers
- OapbContainerId
- OapbContainerPodId
- OapbContainerQual

```

                BPXZOAPB      ,
OAPB      DSECT
OAPB1     DS      CL256
OAPB2     DS      CL20
                ORG      OAPB2
OAPBDEFAULTUSERIDLEN DS X Length of default userid
OAPBDEFAULTUSERID DS CL8 Default userid
OAPBDEFAULTGROUPIDLEN DS X Length default groupid
OAPBDEFAULTGROUPID DS CL8 Default groupid
                DS      CL2      reserved
OAPB3     DS      CL196
OAPB_LEN EQU      *-OAPB
OAPBCOPYONFORK DSECT      These fields will be copied to the
                        child Oapb on fork()
OAPBC1    DS      CL56
OAPBC2    DS      CL24
                ORG      OAPBC2
OAPBUIDS  DS      CL12      User IDs
                ORG      OAPBUIDS
OAPBRUID  DS      F          Real Uid
OAPBEUID  DS      F          Effective Uid
OAPBSUID  DS      F          Saved Uid
OAPBGIDS  DS      CL12      Group IDs
                ORG      OAPBGIDS
OAPBRGID  DS      F          Real Gid
OAPBEGID  DS      F          Effective Gid
OAPBSGID  DS      F          Saved Gid
OAPBC3    DS      CL200
OAPBC4    DS      CL160
                ORG      OAPBC4
OAPBCONTAINERIDENTIFIERS DS CL160 Container identifiers
                ORG      OAPBCONTAINERIDENTIFIERS
OAPBCONTAINERID DS CL64 Container ID
OAPBCONTAINERPODID DS CL64 Container Pod ID
OAPBCONTAINERQUAL DS CL32 Container Qualifier
OAPBC5    DS      CL64
OAPBCOPYONFORK_LEN EQU *-
OAPBCOPYONFORK

```

## BPXZOCVT — Base control block for z/OS UNIX

BPXZOCVT maps addresses of common areas for use by z/OS UNIX subcomponents. Only the following fields are externally documented. All other fields are reserved for IBM use only.

- OcvtkernelReady
- OcvtsuperUser

```

                BPXZOCVT      ,
OCVT      DSECT      ,
                DS      CL60
OCVTFLGS  DS      0FL4      Offset +3C
OCVTFLGSB1 DS      0BL1      Offset +3C
OCVTKERNELREADY EQU X'20'      z/OS UNIX Kernel Ready to accept system calls
                ORG      OCVTFLGSB1+X'00000001'
OCVTFLGSB2 DS      1BL1
OCVTFLGSB3 DS      1BL1
OCVTFLGSB4 DS      1BL1
                ORG      OCVTFLGS+X'00000004'
                DS      CL680
OCVTUIWRWCD S DS      CL8      Offset +2E8

```

## BPXZOTCB — z/OS UNIX extension to the TCB

BPXZOTCB maps z/OS UNIX extensions to the TCB.

Only the following fields are externally documented. All other fields are reserved for IBM use only.

- OtcbThli
- OtcbWLMEToken
- OtcbSigPending
- OtcbOapb

```

BPXZOTCB ,
OTCB      DSECT ,
OTCBID    DS      CL4      EBCDIC ID - OTCB
OTCBSP    DS      X        Subpool number of this OTCB
OTCBLEN   DS      FL3      Length of this OTCB
OTCBPTXL  DS      A        -> pthread parameters
OTCBKSER  DS      A        -> KSER
OTCBMEDCLEAR DS 0CL84      Section of Otcb we clear for medium weight  X
                             processes
OTCBFLAGS DS 0BL4          Compare and swap flg
OTCBFLAGSB1 DS 0B
OTCBINITIALTHREAD EQU X'80' Initial thread of a process
OTCBINKERNELCALL EQU X'40' moved to PPRT
OTCBSLEEP EQU  X'20'      Signal sleep() flag which is checked by      X
                             pause().
OTCBCALLEDKERNEL EQU X'10' At sometime in its life, this thread has      X
                             made a system call /CS
OTCBNOPTLSIR EQU X'08'      Signal is being sent from the ptrace PtlSir  X
                             (Ptrace Signal Interface Routine), so signal X
                             delivery should not deliver the signal to the X
                             PtlSir if ptrace mode is on (we're already X
                             there)
OTCBPROCESSCLEANUP EQU X'04' Process being torn down. /CS
OTCBINTASKTERM EQU X'02'      Thread is in the process of task termination. X
                             Set by BPXRRTRM during task term
OTCBYPASSRACF EQU X'01'      Do not do RACF check in kill() routine
                             ORG  OTCBFLAGSB1+1
OTCBFLAGSB2 DS 0B
OTCBPTEXITONLY EQU X'80'      Thread did XAG exitonly
OTCBTHREADPTEXITED EQU X'40' Marked Ptexited
OTCBTHREADTERM EQU X'20'      Thread in terminated state
OTCBIPT EQU  X'10'          Indicates this thread is or was the Initial  X
                             Pthread Task, used by BPXPRMPC to check for X
                             IPT cleanup
OTCBPROCESSCREATOR EQU X'08' 1=>Indicates the dubbing of this thread      X
                             caused the creation of the process
OTCBCANCELINTR EQU X'04'      Cancel interrupt point
OTCBQUIESCEPOSTED EQU X'02'   This task posted by qut
OTCBDUBNEWPROCESS EQU X'01' 0=>Dub as thread, 1=>Dub as process
                             ORG  OTCBFLAGSB2+1
OTCBFLAGSB3 DS 0B
OTCBATTACHEXEC EQU X'80'      attach_exec in progress
OTCBMULTIPROCCLP EQU X'40' 1=> Lower level processes are to be cleaned X
                             up by this thread
OTCBACTIVEACEEMANAGED EQU X'20' 1=Active ACEE managed by RACF          X
                             (initACEE)
OTCBTOGGLEACEEMANAGED EQU X'10' 1=Toggled ACEE managed by RACF        X
                             (initACEE)
OTCBSAVEDACEEMANAGED EQU X'08' MrPwd saved ACEE managed
OTCBINPROCESSTERM EQU X'04' 1=> When PRTRM is terminating a process.    X
                             Used to tell F.S. Termination when PRTRM is X
                             cleaning up.
OTCBTASKACEEUSP EQU X'02'      USP created by TLS_TASK_ACEE#
OTCBMRPWUIDSET EQU X'01'      OtcbMrPwdUID field set
                             ORG  OTCBFLAGSB3+1
OTCBFLAGSB4 DS 0B
OTCBPSEUDODUBBED EQU X'80'      Thread is a pseudo-dubbed kernel task
OTCBTASKSEC EQU X'40'          Thread called BPX1TLS to build a task level  X
                             Acee
OTCBENCLAVEOWNER EQU X'20'      Thread is an owner of a WLM Enclave
OTCBWLMEMANAGED EQU X'10'      Enclave managed by WLM
OTCBTASKACEEINIT EQU X'08'      InitUsp done for Task Level ACEE
OTCBDUBTASKACEE EQU X'04' 0=>Don't Dub Task Level ACEE 1=>Dub Task      X
                             Level ACEE
OTCBPTCREACEE EQU X'02' 0=> No ACEE propagated on Ptcree 1=> ACEE was X

```

```

                                propagated
OTCBPROCINITACEE EQU X'01' 0=> No INITACEE done during dub 1=> X
                                INITACEE done during dub
                                ORG OTCBFLAGS+4
OTCBTHID DS 0CL8 Thread ID
OTCBPPRT DS 0A -> PPRT
OTCBLIGHTWEIGHT EQU X'80' 1 Light weight thread
                                ORG OTCBPPRT+4
OTCBSEQNO DS 0F Sequence number
OTCBSEQNOHIGHERHALF DS H higher half of seq num
OTCBSEQNOWERHALF DS H lower half of seq num
OTCBSIGFLAGS DS 0BL4 Signal Flags1 that are modified by signal X
                                IRBs. Serialized by Compare & Swap
OTCBSIGFLAGSB1 DS 0B
OTCBSIGDISABLE EQU X'80' Signal Delivery is disabled
OTCBSIGPENDING EQU X'40' Signal pending flag
OTCBTIMERSIGNAL EQU X'20' SIGXCPU or SIGKILL is to be generated by the X
                                syscall layer. Either OtcbSIGXCPU or X
                                OtcbSIGKILL is on. This flag exists for X
                                syscall layer performance.
OTCBSIGNALRM EQU X'10' generate in SC layer
OTCBALRMACTIVE EQU X'08' ALR & setitimer REAL
OTCBIGNRBSTATE EQU X'04' Ignore RB state
OTCBSIGDUMP EQU X'02' Dump for terminating signal
OTCBRAISETIMERIRB EQU X'01' Raise() function from Timer IRB
                                ORG OTCBSIGFLAGSB1+1
OTCBSIGFLAGSB2 DS 0B
OTCBIRBSIGNAL EQU X'80' Signal Checker routine should recheck signals X
                                because one of the signal IRBs may have X
                                changed the signals pending
OTCBPTDELAYIRB EQU X'40' Delay IRB for PTRACE
OTCBSIRDISABLE EQU X'20' Disable invocation of SIR
OTCBCANCELASYNC EQU X'10' Cancellation request state 0 = controlled, 1 X
                                = async
OTCBCANCELDISABLED EQU X'08' Cancellation request type 0=enabled, X
                                1=disabled
OTCBSETSIGDISABLE EQU X'04' Syscall must turn on OtcbSigDisable on X
                                return
OTCBCTWACTIVE EQU X'02' cond_timed_wait (BPX1CTW) is active
OTCBIGNDLVKEY EQU X'01' Ignore Dlv key
                                ORG OTCBSIGFLAGSB2+1
OTCBSIGFLAGSB3 DS 0B
OTCBCANCELPENDING EQU X'80' Cancel pending for thrd
OTCBPTREGSINUSTA EQU X'40' Ptrace regs/PSW are in the Usta
OTCBSIGXCPU EQU X'20' SIGXCPU is to be generated by syscall layer
OTCBSIGKILL EQU X'10' SIGKILL is to be generated by syscall layer
OTCBSIGVTALRM EQU X'08' generate in SC layer
OTCBSIGPROF EQU X'04' generate in SC layer
OTCBALLSIGSBLOCKED EQU X'02' All signals are blocked, the same as if X
                                all bits were on in PpstSigMask
OTCBUDPINKERNEL EQU X'01' In UDP syscall
                                ORG OTCBSIGFLAGSB3+1
OTCBSIGFLAGSB4 DS 0B
OTCBVTALRMACTIVE EQU X'80' setitimer VIRTUAL
OTCBPROFACTIVE EQU X'40' setitimer PROF
OTCBRETURNPPSD EQU X'20'
OTCBCALLRTM EQU X'10' CallRTM done by IR1
OTCBNOIRB EQU X'08' Avoid Irb interrupts
OTCBREDRIVE EQU X'04' IRB redrive is in prog
OTCBCPUTIMEOUT EQU X'02' A terminating signal is to be generated due X
                                to the process time limit being exceeded
OTCBDLVTERM EQU X'01' This thread is terminating due to a X
                                terminating signal
                                ORG OTCBSIGFLAGS+4
OTCBSIR2ID DS F Alarm ID set by STIMERM. Changed by X
                                incrementing at start of alarm() and sleep()
OTCBRACGROUP DS A Pointer to RACF structure to be deleted by X
                                next getgr* call
OTCBRACPASSWD DS A Pointer to RACF structure to be deleted by X
                                next getpw* call
OTCBCOMMREQ DS A Address of communications resource associated X
                                with this task (only valid when there is an X
                                active request)
OTCBPTMULTISTATUS DS B Ptrace multi process mode status word value - X
                                will be one of the WastStopFlag... values
OTCBFLAGS2 DS 0CL3 2nd Set of flags
OTCBFLAGSB1 DS 0B
OTCBSTAXDEFERRED EQU X'80' Stax defer performed
OTCBLUKWKEY0 EQU X'40' User Kernwait caller is KEY 0
OTCBKSERWAITINGF EQU X'20' Is Kser waiting?
OTCBAFFINPGMRUNNING EQU X'10' An IPT/thread affinity program is X
                                running on this thread, do no joblogging

```

OTCBREGSINPPSD EQU X'08'	User regs are in Ppsd at time of Freeze_This_Thread	X
OTCBREGSINIRB EQU X'04'	User regs are in IRB at time of Freeze_This_Thread	X
OTCBTIMEDKERNWAIT EQU X'02'	Task is in Timed Kernwait	
OTCBSLOWPATHSYSCALL EQU X'01'	This is a slow-path syscall. User regs are in USTA	X
ORG OTCBFLAGS2B1+1		
OTCBFLAGS2B2 DS 0B		
OTCBOSENVACTIVE EQU X'80'	Task is active in the osenv	
OTCBOSENVGET EQU X'40'	Task issued osenv_get	
OTCBOSENVWLMJOIN EQU X'20'	Task is joined to a WLM enclave as a result of osenv_set	X
OTCBOSENVSECURITY EQU X'10'	Security environment was saved by osenv	
OTCBCHKPTUNSAFE EQU X'08'	Task is checkpoint unsafe due to being in kernel	X
OTCBINRSTWAIT EQU X'04'	Task is waiting for OMVS to be restarted	
OTCBDORSTWAIT EQU X'02'	Fastpath syscall requests task be put in restart wait	X
OTCBSPBUPDATE EQU X'01'	Used for SPB/IR1 serialization	
ORG OTCBFLAGS2B2+1		
OTCBFLAGS2B3 DS 0B		
OTCBDEFERSIGS EQU X'80'	Defer sigs is in effect	
OTCBLATCHPROBLEM EQU X'40'	Latch Cleanup Problem detected at termination time	X
OTCBF2_ATTACHEDTASK EQU X'20'	This task has done a localspawn or attach	X
OTCBPTHDFORKCHILD EQU X'10'	This child process was created via fork from a pthread	X
ORG OTCBFLAGS2+3		
OTCBALRMGTYEAR DS F	Alarm time in seconds greater than 365 days used by alarm() and sleep() functions	X
OTCBCOFPTR DS A	Address of CopyOnFork area	
OTCBDLVIRB DS A	Address of RB that called Signal Delivery	
OTCBDUBRBSQN DS F	Sequence number of RB that was DUBed	
OTCBREGRBSQN DS F	Sequence number of RB that registered for signals	X
OTCBSPB	the ALET for this SPB is PRIMARY. For BPXJCSA, the ALET is HOME.	X
OTCBSYSCALLCODE DS F	System call number	
OTCBLECB DS A	Ptr to ECB used to wait for a latch to be obtained	X
OTCBPPSDPTR DS 0A	-> PPSD	
OTCBPPSD DS A	-> PPSD	
OTCBCTWID DS F	cond_timed_wait stimerm ID	
OTCBSTACKNONSW DS A	Dynamic stack for Non-space switched syscalls. Only valid when OTCBSYSCALLCODE is non-zero. Contains address of 1st #SAMAP area following RUCA. Addressable in user home space.	X
OTCBOTIM DS A	-> interval timers	
OTCBOAPB DS A	-> OAPB	
OTCBMEDCLEAR2 DS 0CL248	We can't clear Oapb, multiproc quiesce references	X
OTCBPTPICPARMSPTR DS A	Pointer to ptrace recovery environment parameters (PIC parms)	X
OTCBPTEVENTID DS F	Ptrace event ID, that identifies why this thread stopped for ptrace	X
OTCBPTLCLPPSPTR DS A	Ptrace local Ppsd pointer	
OTCBMVSPAUSEECBLIST DS A	Pointer to the BPXZECBL - System copy of user ECB addresses passed to MVSpauseInit	X
OTCBSAVEDSCB DS A	Saved SCB addr of STAI on entry to Local Child Process	X
OTCBUECBLIST DS A	Pointer to the BPXZECBL - System copy of user and system ECBs address for the BPXLUKW - User KernWait service	X
OTCBUIDS DS 0CL12	User IDs for Thread	
OTCBRUID DS F	Real Uid	
OTCBEUID DS F	Effective Uid	
OTCBSUID DS F	Saved Uid	
OTCBSAVEDACEE DS A	MRPWD saved Acee	
OTCBPPRX DS A	Address of the Pprx, an extension of the Pprt	
OTCBMRPWDUID DS F	Password verified UID	
OTCBPSWBYT03 DS F	Caller's PSW bytes 0-4 (Used by JCPR to setup BPXZUSTA)	X
OTCBMRPWDUSERNAME DS CL8	Password verified userid	
OTCBSAVEDSECE NV DS A	Pointer to ACEE saved by BPX1ENV for a toggle request	X
OTCBMVUSERIDPTR DS A	Pointer to userid of this thread, points to either OtcbLoginNInfo or OasbLoginNInfo	X
OTCBLOGINNINFO DS 0CL13	Task userid and length	

```

OTCBLOGINNLEN DS F      Task userid length
OTCBLOGINNAME DS CL9    Tasks userid, must be '00'x (null)          X
                           terminated. Preceding length does not include '
                           ' terminating null
OTCBPRIN2FLAGS DS B     This field is modified by BPXPRIN1, and it is X
                           used by BPXPRIN2. See PPSQ for the mapping X
                           and more details
                           DS      CL2      Reserved, keep word bdy
OTCBTHLI DS      A      -> Thli. This field must never change offsets X
                           within the OtcB since the Thli is an external X
                           control block and the user will have to go X
                           through the OtcB to get to the Thli
OTCBACTSCTBNODEPTR DS A  Active Acee SCTB node ptr, zero if ACEE is X
                           private
OTCBTOGGLEDSCCTBNODEPTR DS A Toggled Acee SCTB node ptr, zero if ACEE X
                           is private
OTCBPAG DS      F      Process Auth Groups
OTCBGIDS DS      0CL12  Group IDs for Thread
OTCBRGID DS      F      Real Gid
OTCBEGID DS      F      Effective Gid
OTCBSGID DS      F      Saved Gid
OTCBRACGIDSPTR DS A     Addr of saved group list
OTCBWLMETOKEN DS BL8    WLM Enclave token associated with the thread
OTCBsavedGID DS F      Gid set by getpwnam, used by setgid
OTCBALIASNINFO DS 0CL13
OTCBALIASNLEN DS F      Task alias length
OTCBALIASNAME DS CL9    Tasks alias, must be '00'x (null) terminated. X
                           Preceding length does not include '
                           ' terminating null
                           DS      CL3      Reserved, keep word bdy
OTCBOSENVTOKEN DS 0CL8  osend environment token
OTCBOSENVCELLPTR DS A   Ptr to osend environment cell element
OTCBOSENVSEQN DS F      seq number associated with the osend cell
OTCBREDRIVETIME DS F    Time to delay signal IRB
OTCBSHLLoaderINFO DS 0CL48 Shared Library data
OTCBSHLLoader1DSPADDR DS A Shared Library loader data1 addr in data X
                           space
OTCBSHLLoader1DSPAGES DS F Shared Library loader data1 len in data X
                           space
OTCBSHLLoader1DSPALET DS F Shared Library loader data1 alet of data X
                           space
OTCBSHLLoader1DSPSTOKEN DS CL8 Shared Library loader data1 stkn of X
                           data space
OTCBSHLLoader2DSPADDR DS A Shared Library loader data1 addr in data X
                           space
OTCBSHLLoader2DSPAGES DS F Shared Library loader data1 len in data X
                           space
OTCBSHLLoader2DSPALET DS F Shared Library loader data1 alet of data X
                           space
OTCBSHLLoader2DSPSTOKEN DS CL8 Shared Library loader data1 stkn of X
                           data space
OTCBSHLLoaderTOK DS CL8 Shared Library loader token len used by X
                           BPXXSHLB INIT
OTCBsmKLATCHFLAGCOUNT DS F OcvtsMKLatchCount incremented
OTCBsavePprt DS A      Addr of Pprt saved during task term when X
                           OtcBpprt is changed so that BPXMIPCE can find X
                           real Pprt of running task
OTCBsysCALLSTART DS BL8 Start Timeused Value for active syscall
OTCBPECBPTR DS A      Prt to a PECB
                           DS      CL24    Reserved for future use
*
*
*   NOTE: The size of this control block is retrieved dynamically
*   during runtime by the modules that need it.
*   When adding additional fields to this control block,
*
*   =====> THE ONLY MODULE THAT *MUST* BE RECOMPILED IS BPXPRIT
*
*
*
OTCB_END DS      0C      Make CB end on doubleword
OTCB_LEN EQU     *-OTCB
*
* *****
* *
* * NOTE: The "OtcBCopyOnFork" is contiguous to the end of the OtcB. *
* * IPCS modules respecify its base on the address of OtcB_End *
* * because the field OtcBcofPtr does NOT contain an address *
* * that is usable by IPCS without another ?ASAXACC. *
* *
* *****
*

```

```

*
OTBCBCOPYONFORK DSECT      These fields will be copied to the child OtcB X
                           on fork()
OTCBSYSCALLENTRYSTATUS DS A Ptr to regs and stuff at entry to the      X
                           syscall layer
OTCBUSTAPTR DS A           Pointer to user status area containing the    X
                           syscall issuer's regs and PSW. (Mapped by    X
                           BPXZUSTA. Used by ptrace.)
OTCBGROUPDBSEARCH DS CL8   Group name for group data base search        X
                           (getgrent)
OTCBUSERDBSEARCH DS CL8   User name for user data base search           X
                           (getpwent)
OTCBSTORAGEFORCPR DS A     Dynamic area for BPXJCPR
OTCBSTORAGEFORCPRKEY DS X Storage key of dynamic area for JCPR
OTCBSTORAGEFORCPRLEN DS FL3 Length of JCPR dynamic area
OTCBSTORAGEFORCPRSP DS X Storage SP of dynamic area
OTCBCOPYFLAGS DS 0B        Copy on Fork Flags
OTCBSETUIDEXEC EQU X'80'   Setuid Exec in progress
OTCBMVSAUTHLIB EQU X'40'   Pgm loaded from MVS authorized library by    X
                           exec/execmvs
OTCBEXECPROCESS EQU X'20' This process image was created by            X
                           exec/execmvs
OTCBSIGPOSTINPC EQU X'10' Sig IRB will post ThliComECB even when a      X
                           Linkage Stack is detected, signal will be    X
                           rescheduled for future delivery
OTCBPOEATTRSET EQU X'08'   The OtcBPOeAttr have been set by __poe
                           ORG OTCBCOPYFLAGS+1
                           DS CL2 Reserved
OTCBDAEMONINFO DS 0CL12    This info will be set and reused to improve  X
                           performance by bypassing multiple RACF calls X
                           for the same info
OTCBsavedUID DS F          Remembered UID
OTCBsavedUSERNAME DS CL8   Remembered UserID
OTCBWLMUSERDATAINFO DS 0CL16 WLM Server information. These fields are   X
                           needed to manage the application data and      X
                           file descriptor list. A single area is        X
                           obtained for both needs. The area is pointed  X
                           to by OtcBWLUserDataPtr. The application      X
                           data is always first in the area followed by  X
                           the FDL area.
OTCBWLMUSERDATAPTR DS A    Address of user data storage
OTCBWLMUSERDATALEN DS F    Length of user data storage -- the entire    X
                           area
OTCBWLMUSERDATAKEY DS X    Key of user data area
                           DS CL3 reserved
OTCBWLMAPPLEN DS F         Length of the application data portion of the X
                           user data area. The file descriptors occupy  X
                           the remainder of the user data area
OTCBSMFBUFFPTR DS A        Address of key 0 copy of user SMF record
OTCBSMFBUFFLEN DS F        Length of key 0 copy of user SMF record
OTCBPOEATTR DS 0CL96       __poe() port of entry info
OTCBPOEPEERIPADDR DS 0CL16 Peer IP Address
OTCBPOEPEERIPV6PREFIX DS CL12
OTCBPOEPEERIPV4ADDR DS F
OTCBPOETERMID DS CL8       TERMINAL Profile Name
OTCBPOELABEL DS CL8        Security Label of poe
OTCBPOEPROFILE DS CL64     SERVAUTH Profile Name
*
* *****
* *
* * NOTE: The size of this control block is retrieved
* * dynamically during runtime by the modules that need it. When*
* * adding additional fields to this control block, ==> THE *
* * ONLY MODULE THAT *MUST* BE RECOMPILED IS BPXPRIT
* *
* *****
*
*
OTBCBCOFEND DS 0C          Make CB end on dword
OTCBCOPYONFORK_LEN EQU *-OTCBCOPYONFORK
*
* *****
* *
* * **** This is end of "OtcBCopyOnFork" area ****
* *
* *****
*
*
* *****
* *
* * The following based area is used by BPXPRGUG and BPXPRSGR. A *

```

```

*      * copy of the last getgroupsbyname will be saved here if done      *
*      * after a __passwd() with the same name.                          *
*      *                                                                    *
*      * *****                                                            *
*
*
OTCBRACGIDS DSECT
OTCBRACGIDSHEADER DS 0CL8
OTCBRACGIDSTOTNUM DS F
OTCBRACGIDSCURNUM DS F
OTCBRACGIDSLIST DS F
OTCB#ID EQU C'OTCB' Control Block Acronym
OTCB#MEDCLEAR2LEN EQU 416 Length to clear
OTCB#LEN EQU 520
*
*                               Length of OTCB
*
OTCB#ONLYLEN EQU 352 Length of OTCB only
OTCB#SP EQU 230 Subpool for the OTCB
OTCBRACGIDS_LEN EQU *-OTCBRACGIDS

```



## Appendix C. Mapping macros—AMODE 64

Mapping macros map the parameter options in many callable services. The fields with the comment "Reserved for IBM Use" are not programming interfaces. A complete list of the options for each macro is listed in the macro in [“Macros mapping parameter options” on page 1055](#).

Most of the mapping macros can be expanded with or without a DSECT statement. The invocation operand DSECT=YES (default) can be used with either reentrant or nonreentrant programs with the appropriate rules governing the storage backed by the USING statement.

Many of the mapping macros exploit the fact that DC expands as a DS in a DSECT and as a DC with its initialized value in a CSECT. When these fields are expanded as or within DSECTs, the program is responsible for initializing the necessary fields.

### Macros mapping parameter options

Specifying DSECT=YES (the default for all macros) creates a DSECT. Addressability requires a USING and a register pointing to storage.

Specifying DSECT=NO (exceptions are listed when this is not allowed) allocates space in the current DSECT or CSECT. In reentrant programs, programmers can place these macros in the DSECT with DSECT=NO, and addressability is accomplished without the individual USING required by DSECT=YES. Nonreentrant programs can place their macros in the program's CSECT and addressability is obtained through the program base register(s).

Specifying LIST=YES (the default for most macros) causes the expansion of the macro to appear in the listing. You can override this by using PRINT OFF.

Specifying LIST=NO removes the macro expansion from the listing.

Additional keywords VARLEN and PREFIX are described in the individual sections where they apply.

### BPXYAIO — Map asyncio parameter list

AMODE 31 callers use [“BPXYAIO — Map asyncio parameter list” on page 1055](#).

```

                SYSSTATE AMODE64=YES
                BPXYAIO      ,
* ----- 64-Bit Version
* -----
** BPXYAIO: Asyncio parameter block
** Used by: AIO
AIO             DS      0D
AIOFD           DS      F      File Descriptor
AIOBUFFDW      DS      0CL8   Eight byte addresses
AIOBUFFALET    DS      F      Alet for AioBuffPtr
&AIOBUFFPTR31  DS      F      Buffer Pointer
AIOBUFFSIZE    DS      F      Buffer Length or Iov count
AIOOFFSETDW    DS      0CL8   Offset in File
AIOOFFSETH     DS      F      Offset in File highword
AIOOFFSET      DS      F      Offset in File lowword
AIOMSGEVENT    DS      0C      Message Event overlays SigEv
&AIOSIGEVENT31 DS      CL20   POSIX Signals
AIOREQPRIO     DS      F      REQUEST PRIORITY
AIOLIOOPCODE   DS      F      LIO_LISTIO() OP
*
                ORG      AIOLIOOPCODE
AIOCMD         DS      F      Command Code
AIONOTIFYTYPE  DS      H      Notification Type
AIOCFLAGS      DS      XL1     Control Flags
AIOOK2COMPIMD  EQU      X'80'  Ok to complete immediately
AIOCALLB4      EQU      X'40'  Call exit before redrive
AIOSYNC        EQU      X'10'  Do synchronously
AIOEXITMODETCB EQU      X'08'  0=SRB, 1=TCB
AIOCANCELNOWAIT EQU      X'04'  Nowait option on cancel

```

```

AIOCANCELNONOTIFY    EQU    X'02'    NoNotify option on cancel
AIOTCBAFFINITY       EQU    X'01'    TCB Affinity I/O
AIOCFLAGS2           DS      XL1      Control Flags2
AIOUSERKEY           EQU    X'F0'    Caller's User's Key bit positions
AIOUSEUSERKEY        EQU    X'08'    Use User's Key for moves
AIOTHLICOEBCB        EQU    X'04'    AioEcbPtr points tp ThliComEcb
AIOCOMMBUFF          EQU    X'02'    Common Area Buffer
AIOMSGIOVALET        DS      F        Alet for recvmsg/sendmsg IOV
AIOIOVBUFFALET       DS      F        Alet for all IOV buffers
*
AIORV                DS      F        Return value
AIORC                DS      F        Return code
AIORSN               DS      F        Reason code
*
AIOPOSIXFLAGS        DS      XL4      Posix flags
&AIOEXITPTR31        DS      F        Pointer to user exit
AIOEXITDATA          DS      CL8      User Data for exit program
AIOECBPTR            DS      F        ECB address
AIOSOCKADDRLEN        DS      F        Sockaddr length
&AIOCKADDRPTR31      DS      F        Sockaddr pointer
AIOTIMEOUT           DS      F        TimeOut Value in Milli-seconds
AIOACEE              DS      F        SRB ACEE for MLS
AIOISICODE           DS      XL2      Signal si_code
AIORES06             DS      CL2      Reserved
AIOLEN               DS      F        (Output,debug) Len of AIO rcvd
AIOENDVER1           DS      0D       End of Original Aiocb
* ----- 64-Bit Extension
AIOLP64              DS      0D
&AIOBUFFPTR64        DS      AD      Buffer Ptr
&AIOEXITPTR64        DS      AD      Exit Program Address
&AIOSIGEVENT64       DS      CL32     SigEvent Structure
&AIOCKADDRPTR64      DS      AD      Sockaddr Ptr
DS      CL8
* ----- Version 3 Extension
AIOLOCSOCKADDRPTR    DS      AD      Local Sockaddr Ptr for ANR
AIOLOCSOCKADDRLEN    DS      F        Local Sockaddr Len for ANR
AIOANRSOCKET         DS      F        Accepted Socket for ANR
DS      CL48
AIOENDVER3           DS      0D       End of Version 3 extension
AIOEND               DS      0D       End of Aiocb
*
AIO#LENGTH           EQU    *-AIO     Length of this structure
*
** AIO command values
AIO#ACCEPT            EQU    126
AIO#CONNECT           EQU    128
AIO#READ              EQU    43
AIO#WRITE             EQU    54
AIO#READV             EQU    133
AIO#WRITEV            EQU    144
AIO#RECV              EQU    134
AIO#SEND              EQU    138
AIO#RECVFROM          EQU    135
AIO#SENDTO            EQU    140
AIO#RECVMSG           EQU    243
AIO#SENDMSG           EQU    244
AIO#ANR               EQU    264
AIO#BRLOCK            EQU    3
AIO#SELPOLL           EQU    2
AIO#CANCEL            EQU    1
*
** AIO notify type
AIO#POSIX             EQU    0
AIO#MVS               EQU    1
AIO#MSGQ              EQU    2
*
** AIO Message Event Structure
*   For AioNotifyType of AIO#MSGQ the AioMsgEvent
*   structure overlays AioSigEvent (31-bit location).
*   Msgbuf and Msgbuf64 are defined in BPXYMSG.
*   IPC_NOWAIT is defined in BPXYIPCP.
*
AIO_BEFORE_MSGEV      DS      0C      Note current position
ORG    AIOMSGEVENT
AIOMSGEV_QID          DS      F        Msg Queue Id
AIOMSGEV_SIZE         DS      H        Length of Msg_mtext
AIOMSGEV_FLAG         DS      H        0 or IPC_NOWAIT
AIOMSGEV_ADDR64       DS      D        Amode(64)-> MsgBuf64
ORG    AIOMSGEV_ADDR64
AIOMSGEV_ADDRH        DS      F
AIOMSGEV_ADDR         DS      F        Amode(31)-> MsgBuf
ORG    AIO_BEFORE_MSGEV    Return to above

```

```

*
AIO#MSGTEXTMAX      EQU  240           Max Msg_MText
*
** AIO Signal Event
SIGEVENT             DSECT ,
SIGEVENT             DS 0F
SIGEV_NOTIFY         DS    F           NOTIFICATION TYPE
SIGEV_SIGNO          DS    F           SIGNAL NUMBER
SIGEV_VALUE          DS    &AIOPTRSIZE SIG VALUE
                     ORG    F
SIVAL_INT            DS    F           SIGEV_VALUE
                     ORG    &AIOPTRSIZE
SIVAL_PTR            DS    &AIOPTRSIZE
SIGEV_NOTIFY_FUNCTION DS    &AIOPTRSIZE NOTIF. FUNCTION
SIGEV_NOTIFY_ATTRIBUTES DS &AIOPTRSIZE NOTIF. ATTRIBUTES
*
SIGEV#LENGTH         EQU  *-SIGEVENT   Length of this structure
*
*   SIGEV_NOTIFY Values
SIGEV_SIGNAL          EQU    0         GENERATE A SIGNAL
SIGEV_NONE            EQU    1         DON'T GENERATE SIGNAL
SIGEV_THREAD          EQU    2         Call Notif. function
*
** AIOTIMEOUT VALUES
AIO#FOREVER           EQU    0         NO TIMEOUT, JUST WAIT
AIO#NOWAITING         EQU    X'FFFFFFF' NO WAITING, JUST CHECK
** AIO CANCEL RETURN VALUES
AIO_CANCELED          EQU    1         ALL CANCELS SUCCESSFUL
AIO_NOTCANCELED       EQU    2         AT LEAST 1 CANCEL FAILED
AIO_ALLDONE           EQU    3         NONE CANCELED, ALL COMP
*
** BPXYAIO End

```

## BPXYCCA — Map input/output structure for \_\_console()

AMODE 31 callers use “BPXYCCA — Map input/output structure for \_\_console()” on page 1060.

```

                SYSSTATE AMODE64=YES
                BPXYCCA
** BPXYCCA:  Msg Attributes for console_np service
** Used By:  CCS
CCA             DSECT ,
CCABEGIN       DS    0D
*
CCAVERSION     DC    AL2(CCA#VER)
*
CCARES01       DS    CL2    Reserved
CCAMSGLENGTH   DS    F      Length of msg pointed to by CCAMSGPTR
CCAMSGPTR       DS    AD     Pointer to Msg text
CCARES02       DS    CL4    Reserved
CCAENDVER1     DS    0F     End of Version 1
CCASTARTVER2   DS    0F     Start of Version 2
CCARES03       DS    F      Reserved
CCAWTOPARMS    DS    0F     Start of WTO message attributes
CCAROUTCDELIST DS    AD     Pointer to list of message routing codes X
CCADESCLIST    DS    AD     Pointer to list of message descriptor codes X
CCAWMCSFLAGS   DS    0F     WTO MCS Flags
CCAMCSFLAGB1   DS    XL1    MCS flags byte 1
CCAHRDCPY      EQU    X'80' Send message to hard copy log only
CCAMCSFLAGB2   DS    XL1    MCS flags byte 2
CCAMCSFLAGB3   DS    XL1    MCS flags byte 3
CCAMCSFLAGB4   DS    XL1    MCS flags byte 4
CCAWTOTOKEN    DS    F      Token for message to be issued
CCAMSGIDPTR    DS    AD     Pointer to location where message is stored by BPX1CCS X
CCARES07       DS    F      Reserved
CCADOMPARMS    DS    0F     Delete message parameters
CCADOMTOKEN    DS    F      Token of message(s) to be deleted
CCAMSGIDLIST   DS    AD     Pointer to list of message ids to be deleted X
CCAENDVER2     DS    0D     End of version 2
CCASTARTVER3   DS    0CL40  Start of version 3
CCAMODCARTPTRG DS    AD     Pointer 8 byte CART returned for MODIFY/STOP command X
CCAMODCONSOLEIDPTRG DS AD Pointer to 4 byte ConsoleID returned X

```

CCAMSGCART	DS	CL8	for MODIFY/STOP command Supplied - CART to be specified on WTO when message is issued	X
CCAMSGCONSOLEID	DS	CL4	Supplied - ConsoleID to be specified on WTO when message is issued	X
CCARES08	DS	CL12	Reserved	
CCAENDVER3	DS	OD	End of Version 3	
* Constants				
CCA#VER	EQU	CCA#VER02	Current version	
CCA#VER01	EQU	1	Version 1 of this structure	
CCA#VER02	EQU	2	Version 2 of this structure	
CCA#VER03	EQU	3	Version 3 of this structure	
CCA#LENGTH	EQU	*-CCABEGIN		X
			Length of CCA	
CCA#VER1LEN	EQU	CCAENDVER1-CCABEGIN		X
			Length of Version 1 CCA	
CCA#VER2LEN	EQU	CCAENDVER2-CCABEGIN		X
			Length of Version 2 CCA	
CCA#VER3LEN	EQU	CCAENDVER3-CCABEGIN		X
			Length of Version 3 CCA	
** BPXYCCA End				

## BPXYDCOR – dbx cordump cache information

BPXYDCOR contains the mapping of dump related information used by dbx when a dump is being formatted. AMODE 31 callers use “BPXYDCOR – dbx cordump cache information” on page 1069.

```

                BPXYDCOR    PARMSG=YES
*
* *****
* * Level information
* *
* *****
*
*
DCOR_LEVEL1 EQU 65536    65536='00010000'x.
DCOR_LEVEL2 EQU 131072  131072='00020000'x.
*
* *****
* * Function codes for BPXGMCDE routine
* *
* *****
*
DCOR_OPEN# EQU 1
DCOR_CLOSE# EQU 2
DCOR_STATUS# EQU 3
*
* *****
* * Open return codes
* *
* *****
*
DCOR_CDERC_OK EQU 0      The specified function completed successfully
DCOR_CDERC_PARMERR EQU 4 A parameter error was detected. See return value 1 for more detail X
DCOR_CDERC_PROCERR EQU 8 A DCORE processing error occurred. See return value 1 for more detail X
DCOR_CDERC_IKJTSOEVERR EQU 12 An error was encountered trying to establish a TSO environment with the IKJTSOEV service. See return values for more information X
DCOR_CDERC_IKJEFTSRERR EQU 16 An error was encountered trying to run the REXX EXEC with the IKJEFTSR service. See return values for more information X
DCOR_CDERC_ALLOCATEERR EQU 20 An error was encountered trying to allocate one of the user specified data sets. X
DCOR_CDERC_IRXINITERR EQU 28 An error was encountered trying to establish a REXX environment X
*
* *****

```

```

*      *
*      * Status return codes
*      *
*      *****
*
*
DCOR_CDERC_STATUS_OPENCOMPLETE EQU 0
DCOR_CDERC_STATUS_OPENCONTINUING EQU 1
DCOR_CDERC_STATUS_OPENTERMINATED EQU 2
DCOR_CDERC_STATUS_INVALIDTOKEN EQU 3
*
*      *****
*      *
*      * Status Rc values when Status return code is
*      * Dcor_CDErc_Status_OpenContinuing
*      *
*      *****
*
*
DCOR_STATUS_CONT_STARTTSOENV EQU 0 Starting the TSO environment
DCOR_STATUS_CONT_EXECSTARTED EQU 1 BPXTIPCS started
DCOR_STATUS_CONT_EXECCLIST EQU 2 BPXTIPCS allocating CLIST data set
DCOR_STATUS_CONT_DUMPDDIR EQU 3 BPXTIPCS allocating/creating dump      X
        directory via BLSCDDIR
DCOR_STATUS_CONT_ALLOCDUMPDS EQU 4 BPXTIPCS allocating the dump data    X
        set
DCOR_STATUS_CONT_INVOKEIPCS EQU 5 BPXTIPCS invoking IPCS
DCOR_STATUS_CONT_INVOKEVERBX EQU 6 BPXTIPC2 invoking VERBX routine
DCOR_STATUS_CONT_ANALYSISSTART EQU 7 Dump analysis started
DCOR_STATUS_CONT_ANALYSISPROCASIDS EQU 8 Analysis processing Asids
DCOR_STATUS_CONT_EXECEXITING EQU 9 BPXTIPCS exiting
DCOR_STATUS_CONT_RECALL EQU 10 BPXTIPCS recalling data set
*
*      *****
*      *
*      * R1 values when return code is Dcor_CDErc_ParmErr
*      *
*      *****
*
*
DCOR_R1_PARMERR_DUMPDSNREQ EQU 1 The name of a dump data set is      X
        required
DCOR_R1_PARMERR_HFSDSNREQ EQU 2 The name of a dump data set in the HFS X
        could not be found
*
*      *****
*      *
*      * R1 values when return code is Dcor_CDErc_ProcErr
*      *
*      *****
*
*
DCOR_R1_PROCErr_SYSTEMERRATC EQU 1 An unexpected system error has    X
        occurred while trying to establish the IPCS      X
        environment. The R2 value contains an ABEND      X
        reason code
*
*      *****
*      *
*      * R1 values when return code is Dcor_CDErc_AllocateErr
*      *
*      *****
*
*
DCOR_R1_ALLOCATEERR_LOGDSN EQU 1 Error allocating the log data set.    X
        The R2 field is the return code from            X
        allocation and the R3 field is the reason        X
        code.
DCOR_R1_ALLOCATEERR_EXECDSN EQU 2 Error allocating the EXEC data set.  X
        The R2 field is the return code from            X
        allocation and the R3 field is the reason        X
        code.
*
*      *****
*      *
*      * Function codes for BPXGMPTR Ptrace Dump Access Routine
*      *
*      *****
*
*
DCOR_ASID_LIST# EQU 1
DCOR_SET_ASID# EQU 2

```

```

DCOR_PID_LIST# EQU 3
DCOR_SET_PID# EQU 4
DCOR_LDINFO# EQU 5
DCOR_THREAD_LIST# EQU 6
DCOR_THREAD_CURRENT# EQU 7
DCOR_SET_THREAD# EQU 8
DCOR_PSW# EQU 9
DCOR_GPR_LIST# EQU 10
DCOR_THREAD_STATUS# EQU 11
DCOR_READ_D# EQU 12
DCOR_ERROR_PSW# EQU 13
DCOR_CAPTURE# EQU 14
DCOR_ERROR_GPR_LIST# EQU 15
DCOR_FLT_LIST# EQU 16
DCOR_ERROR_FLT_LIST# EQU 17
DCOR_CONDINFO# EQU 18
DCOR_IPCSCMD# EQU 19
DCOR_PTRRC_OKVALUE EQU 0 The specified function completed successfully
DCOR_PTRRC_ASIDNOTFOUND EQU 1 The requested asid(s) not in dump
DCOR_PTRRC_ASIDNOTSET EQU 2 An ASID or PID has not been established      X
    for this session
DCOR_PTRRC_REQTYPE_NOTDEFINED EQU 3 The function type provided on this  X
    request is not supported by BPXGMPT2
DCOR_PTRRC_REQINVALIDTOKEN EQU 4 The open token provided on this        X
    request is not valid
DCOR_PTRRC_REQDCORTERMINATED EQU 5 Dcor dump access services are not    X
    available
DCOR_PTRRC_THREADNOTFOUND EQU 6 The request thread(s) were not in the  X
    dump
DCOR_PTRRC_THREADNOTSET EQU 7 The current thread has not been           X
    established
DCOR_PTRRC_PIDNOTSET EQU 9 The request PID(s) were not in the dump
DCOR_PTRRC_PIDNOTFOUND EQU 10 The current process has not been          X
    established
DCOR_PTRRC_STORAGE_NOTINDUMP EQU 11 The requested storage was not       X
    dumped
DCOR_PTRRC_NASTANDALONEDUMP EQU 12 Not supported in a standalone dump
DCOR_PTRRC_ABEND_OCCURRED EQU 13 Not supported in a standalone dump
DCOR_PTRRC_STORAGE_LENGTH_BAD EQU 14 The requested storage length was   X
    zero
DCOR_PTRRC_SOME_STORAGE_IN_DUMP EQU 15 The number of bytes of storage    X
    successfully retrieved is returned in the reason code field          X

RSNOKVALUE EQU 0
RSNDCORERROR EQU 1      See Dcor return codes
RSNMVSError EQU 2      Usually an out of storage condition or an        X
   abend
RSNIPCSERROR EQU 3      When An IPCS error occurs use the DCOR log to X
    view the messages generated by IPCS (normally suppress)

RSNCSVERROR EQU 4
RSNCSVMODI12ERR EQU 1
RSNCSVMODI3ERR EQU 2
RSNCSVTOOMANYEXTENTS EQU 3
*
* *****
* * parameter definitions for BPXGMPT2 Ptrace Dump Access Routine *
* * 1. Parm 1 function code *
* * 2. Params 2 Token returned from DCOR_OPEN# *
* * 3. Params 3-5 Function parameters *
* * 3. Params 6-8 retvalue, retcode, rsncode *
* *****
*
PARMS DSECT
PARMS_FUNCYPEPTR DS 1AL4
PARMS_DCOMTOKENPTR DS 1AL4
PARMS_INTERFACE DS 0CL0012
    ORG PARMS_INTERFACE
PARMS_CAPTURE DS 0CL0012
PARMS_CAPTURE_PSTORADR DS 1AL4
PARMS_CAPTURE_PSTORLEN DS 1AL4
PARMS_CAPTURE_PDAAADR DS 1AL4 Address output buffer
    ORG PARMS_INTERFACE
PARMS_READD DS 0CL0012
PARMS_READD_PSTORADR DS 1AL4
PARMS_READD_PSTORLEN DS 1AL4
PARMS_READD_PDAAADR DS 1AL4 user provided buffer
    ORG PARMS_INTERFACE
PARMS_LDINFO DS 0CL0004
PARMS_LDINFO_OUTBUFPTR DS 1AL4
    ORG PARMS_INTERFACE

```

```

PARMS_THREADLIST DS 0CL0008
PARMS_THREADLIST_OUTBUFPTR DS 1AL4
PARMS_THREADLIST_OUTBUFCNT DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_PIDLIST DS 0CL0008
PARMS_PIDLIST_OUTBUFPTR DS 1AL4
PARMS_PIDLIST_OUTBUFCNT DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_ASIDLIST DS 0CL0008
PARMS_ASIDLIST_OUTBUFPTR DS 1AL4
PARMS_ASIDLIST_OUTBUFCNT DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_THREADCURRENT DS 0CL0004
PARMS_THREADCURRENT_OUTBUFPTR DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_SETASID DS 0CL0004
PARMS_SETASID_INBUFPTR DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_SETPID DS 0CL0004
PARMS_SETPID_INBUFPTR DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_SETTHREAD DS 0CL0004
PARMS_SETTHREAD_INBUFPTR DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_PSW DS 0CL0004
PARMS_PSW_OUTBUFPTR DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_ERROR_PSW DS 0CL0004
PARMS_ERROR_PSW_OUTBUFPTR DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_THREADSTATUS DS 0CL0008
PARMS_THREADSTATUS_OUTBUFPTR DS 1AL4
PARMS_THREADSTATUS_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_GPRLIST DS 0CL0008
PARMS_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_GPRLIST_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_ERROR_GPRLIST DS 0CL0008
PARMS_ERROR_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_GPRLIST_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_FLTLIST DS 0CL0008
PARMS_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_FLTLIST_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_ERROR_FLTLIST DS 0CL0008
PARMS_ERROR_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_FLTLIST_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_CONDITIONINFO DS 0CL0008
PARMS_CONDITIONINFO_OUTBUFPTR DS 1AL4
PARMS_CONDITIONINFO_OUTBUFLN DS 1AL4
    ORG    PARMS_INTERFACE
PARMS_IPCSCMD DS 0CL0012
PARMS_IPCSCMDTEXT_INBUFPTR DS 1AL4
PARMS_IPCSCMDTEXT_INBUFLN DS 1AL4
PARMS_IPCSCMDPRNT_LRECL DS 1AL4
PARMS_XRVPTR DS 1AL4      Return Value
PARMS_XRCPTR DS 1AL4      Return Code
PARMS_XRSNPTR DS 1AL4     Reason Code
PARMS_LEN EQU *-PARMS
PARMSG DSECT
PARMS_FUNCYPEPTRG DS 1AD
PARMS_DCOMTOKENPTRG DS 1AD
PARMS_INTERFACEG DS 0CL0024
    ORG    PARMS_INTERFACEG
PARMS_CAPTUREG DS 0CL0024
PARMS_CAPTURE_PSTORADRG DS 1AD
PARMS_CAPTURE_PSTORLENG DS 1AD
PARMS_CAPTURE_PDATAADRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_READDG DS 0CL0024
PARMS_READD_PSTORADRG DS 1AD
PARMS_READD_PSTORLENG DS 1AD
PARMS_READD_PDATAADRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_LDINFOG DS 0CL0008
PARMS_LDINFO_OUTBUFPTRG DS 1AD
    ORG    PARMS_INTERFACEG
PARMS_THREADLISTG DS 0CL0016
PARMS_THREADLIST_OUTBUFPTRG DS 1AD

```

```

PARMS_THREADLIST_OUTBUFCNTG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_PIDLISTG DS 0CL0016
PARMS_PIDLIST_OUTBUFPTRG DS 1AD
PARMS_PIDLIST_OUTBUFCNTG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_ASIDLISTG DS 0CL0016
PARMS_ASIDLIST_OUTBUFPTRG DS 1AD
PARMS_ASIDLIST_OUTBUFCNTG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_THREADCURRENTG DS 0CL0008
PARMS_THREADCURRENT_OUTBUFPTRG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_SETASIDG DS 0CL0008
PARMS_SETASID_INBUFPTRG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_SETPIDG DS 0CL0008
PARMS_SETPID_INBUFPTRG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_SETTHREADG DS 0CL0008
PARMS_SETTHREAD_INBUFPTRG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_PSWG DS 0CL0008
PARMS_PSW_OUTBUFPTRG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_ERROR_PSWG DS 0CL0008
PARMS_ERROR_PSW_OUTBUFPTRG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_THREADSTATUSG DS 0CL0016
PARMS_THREADSTATUS_OUTBUFPTRG DS 1AD
PARMS_THREADSTATUS_OUTBUFLENG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_GPRLISTG DS 0CL0016
PARMS_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_GPRLIST_OUTBUFLENG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_ERROR_GPRLISTG DS 0CL0016
PARMS_ERROR_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_GPRLIST_OUTBUFLENG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_FLTLISTG DS 0CL0016
PARMS_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_FLTLIST_OUTBUFLENG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_ERROR_FLTLISTG DS 0CL0016
PARMS_ERROR_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_FLTLIST_OUTBUFLENG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_CONDITIONINFOG DS 0CL0016
PARMS_CONDITIONINFO_OUTBUFPTRG DS 1AD
PARMS_CONDITIONINFO_OUTBUFLENG DS 1AD
    ORG  PARMS_INTERFACEG
PARMS_IPCSCMDG DS 0CL0024
PARMS_IPCSCMDTEXT_INBUFPTRG DS 1AD
PARMS_IPCSCMDTEXT_INBUFLENG DS 1AD
PARMS_IPCSCMDPRNT_LRECLG DS 1AD
PARMS_XRVPTRG DS 1AD
PARMS_XRCPTRG DS 1AD
PARMS_XRSNPTRG DS 1AD
PARMSG_LEN EQU *-PARMSG
ASIDLIST_MAP DSECT
ASID_NEXTOFF DS 1FL4      Offset to the next ASID in DcomAsidList
ASID_NUM DS 1FL2
ASID_CPU DS 1FL1          CPUID
ASID_FLAGS DS 0BL1        Status flags
ASID_HOME EQU X'80'        Current HOMEAsid
ASID_PRIM EQU X'40'        Current PRIMARY ASID
ASID_SEC EQU X'20'         Current SECONDARY ASID
    ORG  ASID_FLAGS+X'00000001'
ASID_JOBNAME DS 1CL0009
    DS 1CL0003             Reserved
ASID_PIDCNT DS 1FL4        Number of Pids in this Asid
ASID_ASCB DS 1AL4          Pointer to ASCB
    DS 1CL0004             Reserved
ASID_PIDLISTPTR DS 1AL4    Pointer to the pidlist for This Asid
    DS 1CL0004             Reserved
ASID_MAPEND DS 0C          end of block
ASIDLIST_MAP_LEN EQU *-ASIDLIST_MAP
PIDLIST_MAP DSECT
PID_NEXTOFF DS 1FL4        Offset to the next Pid in DcomPidList
PID_ DS 1FL4               Process id
PID_ASID DS 1FL2           Asid of this Pid

```



```

PID_THIDCNT DS 1FL2      Count of thids in this pid
PID_FOCUSTHREAD DS 1CL0008 Ptrace focus thread
PID_ERRORTHREAD DS 1CL0008 Ptrace error thread
PID_LOGINNAME DS 1CL0009 Tso logon
                DS 1CL0003 Reserved
PID_THIDLISTPTR DS 1AL4   list info for each THID
                DS 1CL0004 Reserved
PID_PENDINGSIGMASK DS 1BL8 Signals pending at the process that could   X
                        not be delivered to any thread
PID_BLOCKEDSIGMASK DS 1BL8 Signals blocked on all thread
PID_MAPEND DS 0C         end of block
PIDLIST_MAP_LEN EQU *-PIDLIST_MAP
CONDINFO_MAP DSECT
COND_CURABENDINFO DS 0CL0016 If current task abended
COND_CURINTCODE DS 1FL2   Interrupt code
COND_CURSIGNUMBER DS 1FL2 Signal number raised
COND_CURABENDCODE DS 0BL4 Abend code
COND_CURABENDFLAGS DS 1BL1 System or user
COND_CURABENDCC DS 1BL3   Abend Number
COND_CURABENDREASON DS 1BL4 Abend Reason
COND_CURILC DS 1FL2       Instruction length
                DS 1CL0002 Reserved
CONDINFO_MAPEND DS 0C     end of block
CONDINFO_MAP_LEN EQU *-CONDINFO_MAP

```

## BPXYINHE – Spawn inheritance structure

AMODE 31 callers use “BPXYINHE – Spawn Inheritance Structure” on page 1084.

```

                SYSSTATE AMODE64=YES
                BPXYINHE
** BPXYINHE: Inheritance Area
** Used By: spawn() callable service
INHE            DSECT
INHEBEGIN      DS 0D
*
INHEEYE        DC C'INHE' Eye catcher
INHELENGTH     DC AL2(INHE#LENGTH) X
                Length of this structure
INHEVERSION    DC AL2(INHE#VER)
INHE#VER       EQU 3      Version of this structure
INHEFLAGS      DS 0BL4   Flags indicating contents of structure
INHEFLAGS0     DS XL1    1st byte
INHESETPGROUP  EQU X'80' Set Process Group using INHEPGROUP
INHESETSIGMASK EQU X'40' Set Signal Mask using INHESIGMASK
INHESETSIGDEF  EQU X'20' Set Signal Defaults using INHESIGDEF
INHESETTCPGRP  EQU X'10' Set TTY Pgrp using INHECTLTTFD
INHESETCWD     EQU X'08' Set CWD using INHECWDPTR
INHESETUMASK   EQU X'04' Set UMASK using INHEUMASK
INHESETUSERID  EQU X'02' Set Userid using INHEUSERID
INHESETREGIONSZ EQU X'01' Set Region using INHEREGIONSZ
INHEFLAGS1     DS XL1    2nd byte
INHESETTIMELIMIT EQU X'80' Set Timelimit with INHETIMELIMIT
INHESETACCTDATA EQU X'40' SET ACCTDATA using INHEACCTDATA
INHESETJOBNAME EQU X'20' SET JOBNAME using INHEJOBNAME
INHEMUSTBELOCAL EQU X'10' Spawn locally or else fail
INHESETDEBUGENV EQU X'08' Setup Debug Environment
INHESETMEMLIMIT EQU X'04' Set MemLimit with INHEMEMLIMIT
INHEFLAGS2     DS XL1    3rd byte
INHEFLAGS3     DS XL1    4th byte
INHEPGROUP     DS F      Process Group for child
INHE#NEWPGROUP EQU 0      Put child in a new proc grp of its own
INHESIGMASK     DS BL8    Signal Mask for child
INHESIGDEF      DS BL8    Set of default signals for child
INHECTLTTFD     DS F      Cntl TTY FD for tcsetgrp() in child
                DS 0F      64-Bit Addressing Version
INHERES01       DS F      Reserved
INHECWDLEN      DS H      Length of the users CWD
INHEACCTDATALEN DS H      LENGTH OF THE USERS ACCTDATA
INHERES02       DS F      Reserved
INHEUMASK       DS XL4    Users Umask
INHEUSERID      DS CL8    New A.S. user identity
INHEJOBNAME     DS CL8    New A.S. jobname
INHEREGIONSZ    DS F      New A.S. region size
INHETIMELIMIT   DS F      New A.S. Time limit
INHERES03       DS F      Reserved
INHECWDPTR      DS AD     Pointer to the users CWD
INHEACCTDATAPTR DS AD     Pointer to the users ACCTDATA

```

```

INHEMMLIMIT      DS      D      New A.S. Memlimit #bytes
INHE#LENGTH      EQU     *-INHE
** BPXYINHE End

```

## BPXYIOC6 – Map IPV6 prerouter structures

BPXYIOC6 is used by transport providers. DSECT= is allowed but ignored. AMODE 31 callers use “BPXYIOC6 – Map IPV6 prerouter structures” on page 1094.

```

        SYSSTATE AMODE64=YES
        BPXYIOC6
NETCONFHDR      DSECT ,
* ----- 64-Bit Version @P2A
NCHHEYECATCHER  DS      CL4      Eye catcher @P2A
NCHIOCTL        DS      F        Ioctl being processed (RAS) @P2A
NCHNUMENTRYRET  DS      F        Number of HomeIF returned via @P2A*
                                SIOCGHOMEIF6 or number of @P2A*
                                GRT6RtEntry's returned via @P2A*
                                SIOCGRT6TABLE. @P2A
NCHBUFFERLENGTH DS      F        Buffer Length @P2A
NCHBUFFERPTR    DS      D        64-bit Buffer Pointer @P2A
NETCONFHDR#LENGTH EQU     *-NETCONFHDR Length of NETCONFHDR
*
*****
* HomeIf Structure *
*****
*
HOMEIF          DSECT ,      HomeIf structure
HomeIfAddress   DS      CL16   Home Interface Address
*
HomeIf#LENGTH   EQU     *-HOMEIF Length of HOMEIF
*
*****
* GRT6RtEntry Structure *
*****
*
GRT6RTENTRY     DSECT ,      GRT6RtEntry Structure
*
GRT6DESTINATION DS      CL16   Destination IP Address
GRT6GATEWAY      DS      CL16   First HOP on the trip if going through *
                                a gateway
GRT6DESTPREFIXLEN DS      F      Destination's Prefix Length which is a *
                                decimal value that specifies how many *
                                of the leftmost contiguous bits of the *
                                address comprise the prefix
GRT6RTMETRIC     DS      F      Metric - hop count. Currently Tcp/Ip *
                                returns 1 for indirect routes and 0 *
                                for direct routes. If route is from *
                                routing daemon, metric is whatever *
                                routing daemon set it to.
GRT6RTFLAGS      DS      F      IPV6 Route Flags.
*
GRT6RTENTRY#LENGTH EQU     *-GRT6RTENTRY Length of GRT6RTENTRY
*
*****
* RT6Entry Structure *
*****
*
RT6ENTRY        DSECT ,      Rt6Entry Structure
*
RT6DESTINATION  DS      CL28   Destination IP address (in an IPV6 *
                                sockaddr structure)
RT6GATEWAY      DS      CL28   First HOP on the trip if going *
                                through a gateway (in an IPV6 *
                                sockaddr structure)
RT6DESTPREFIXLEN DS      F      Destination's Prefix Length, *
                                which is a decimal value *
                                that specifies how many of *
                                the leftmost contiguous *
                                bits of the address *
                                comprise the prefix.
RT6METRIC       DS      F      Metric - hop count *
                                Currently Tcp/IP returns *
                                1 for indirect route and *
                                0 for direct route. *
                                If route is from routing *

```

```

                                daemon, metric is whatever          *
                                routing daemon set it to.
RT6FLAGS          DS      F      IPV6 Route Flags.
*
RT6ENTRY#LENGTH   EQU      *-RT6ENTRY   Length of RT6ENTRY
*
*****
* GRT6RtEntryV2 Structure
*****
*
GRT6RTENTRYV2     DSECT ,          New Route Entry used with DCR A846 - *
                                Route Modification
*
GRT6OLDRTENTRY    DS      CL44      Old GRT6 Route Entry
GRT6RTHOMEIFIDX   DS      F          Route's Home Interface Idx
GRT6RTIFINDEX     DS      F          Route's Interface Index
GRT6RTMTU         DS      H          Route's MTU Value
*
*                   DS      H          Reserved
*                   DS      F          Reserved
*                   DS      F          Reserved
*
GRT6RTENTRYV2#LENGTH EQU      *-GRT6RTENTRYV2   Length of GRT6RTENTRYV2
*
*****
* RT6EntryV2 Structure
*****
*
RT6ENTRYV2        DSECT ,          New Route Entry Used with A846      *
                                MSADDR6V2/MSDELRT6V2 IOCTls
*
RT6OLDENTRY       DS      CL68      Old Route Entry used before A846    *
                                with SIOCMSADDR6/SIOCMSDELRT6 IOCTL
RT6RTHOMEIFIDX    DS      F          Route's Home Interface Idx
*
*                   DS      F          Reserved
*                   DS      F          Reserved
*                   DS      F          Reserved
*                   DS      F          Reserved
*
RT6ENTRYV2#LENGTH EQU      *-RT6ENTRYV2   Length of RT6ENTRYV2
*
*****
* IPV6RtFlags Structure
*****
*
IPV6RTFLAGS       DSECT ,          IPV6RtFlags Structure
*
IPV6FLGR0UTETYPE  DS      XL1      Route Type                          @D1C
IPV6FLGBYTE2      DS      XL1      Reserved
IPV6FLGBYTE3      DS      XL1      Reserved
IPV6FLGBYTE4      DS      XL1      FLAGS:
*
*                   EQU      X'80'   Reserved
*                   EQU      X'40'   Reserved
*                   EQU      X'20'   Reserved
IPV6BITL00PBACk   EQU      X'10'   1 = Loopback Interface
IPV6BITHOME       EQU      X'08'   1 = Home interface
IPV6BITHOST       EQU      X'04'   1 = Host Route. 0 = Network Route
IPV6BITGATE       EQU      X'02'   1 = Gateway
IPV6BITRTUP       EQU      X'01'   1 = Route is active
*
* *-----*
* * SiocGifConf6 - Get IPv6 Interface Configuration.                @D3A*
* *
* * Net_IfConf6Header is passed as the argument of the ioctl and    *
* * is returned with the number of entries and entry length of the  *
* * Net_IfConf6Entry structs that were written to the output buffer.*
* *
* * If Buflen=0=Buffer a Query function is performed and the        *
* * header is returned with: (1) the maximum supported version,    *
* * (2) the total number of entries that would be output and       *
* * (3) the length of each individual entry.
* *
* * If a call to get information fails with RC=ERANGE or with        *
* * (RC=EINVAL & Nif6h_Version is changed) the call is converted  *
* * into a Query function and the content of the output buffer      *
* * is unpredictable.
* *
* * For information on the data returned in this structure refer    *
* * to the z/OS Communication Server's IP Configuration Guide and  *
* * IPv6 Network and Application Design Guide.
* *
* *-----*
NET_IFCONF6HEADER DSECT   Header                                     @D3A

```

```

NIF6H_VERSION DS F      Input for Get IfConf6 Output for Query
NIF6H_ENTRIES DS F      Output: number of entries returned in output *
                          buffer
NIF6H_ENTRYLEN DS F      Output: length of an entry
NIF6H_BUFLen DS F        Input: length of buffer
NIF6H_BUFFER64 DS 0CL8   Input: Amode(64) Buffer ptr
NIF6H_BUFFER64H DS F
NIF6H_BUFFER DS A        Input: Amode(31) Buffer ptr to output buffer *
                          that will be filled with an array of *
                          Net_IfConf6Entrys.
NET_IFCONF6HEADER_LEN EQU *-NET_IFCONF6HEADER
*
NET_IFCONF6ENTRY DSECT   Entry                                     @D3A
NIF6E_NAME DS CL16       x00 interface name (blank padded - no null)
NIF6E_STACKNAME DS CL8   x10 tcpip stack name (blank padded - no null)
NIF6E_ADDR DS CL28       x18 Sock_Inet6_SockAddr of the interface
NIF6E_ROUTE METRIC DS F   x34 route metric
NIF6E_PREFIXLEN DS H     x38 routing prefix length
NIF6E_PREFIXORIGIN DS X  x3A prefix origin, see below
NIF6E_STATUS DS X        x3B status, see below
NIF6E_FLAGS DS 0BL4      x3C Flags:
NIF6E_FLAGS1 DS B
NIF6E_FLAGS2 DS B
NIF6E_FLAGS3 DS 0B
NIF6E_VIRTUAL EQU X'40'
NIF6E_MULTIPPOINT EQU X'08'
NIF6E_MULTICASTCAPABLE EQU X'04'
                          ORG NIF6E_FLAGS3+1
NIF6E_FLAGS4 DS 0B
NIF6E_POINT2POINT EQU X'10'
NIF6E_LOOPBACK EQU X'08'
NIF6E_ONLINK EQU X'01'
                          ORG NIF6E_FLAGS+4
NIF6E_MTU DS F           x40 mtu
*
* *****
* *
* * Constants for nif6h_version                                     @D3A
* *
* *****
*
*
NIF6H#VER EQU 1          Current Version
NIF6H#VER1 EQU 1         Initial Version
*
* *****
* *
* * Constants for nif6e_prefixorigin                               @D3A
* *
* *****
*
*
NIF6H#WELLKNOWN EQU 1
NIF6H#MANUAL EQU 2
NIF6H#RTRADV EQU 3
NIF6H#OTHER EQU 8
*
* *****
* *
* * Constants for nif6e_status                                       @D3A
* *
* *****
*
*
NIF6H#PREFERRED EQU 1
NIF6H#DEPRECATED EQU 2
NIF6H#INVALID EQU 3
NIF6H#INACCESSIBLE EQU 4
NIF6H#UNKNOWN EQU 5
NIF6H#TENTATIVE EQU 6
NIF6H#DUPLICATE EQU 7
NET_IFCONF6ENTRY_LEN EQU *-NET_IFCONF6ENTRY
*                               End SiocGifConf6 ----- @D3A
*
* *****
* *
* * Constants                                                         *
* *
* *****
*
*
IOC6_#HOMEIFPREFIXLEN EQU 128 The prefix length for a home interface *
```

```

                                address returned on the SIOCGHOMEIF6 IOCTL.
IOC6_NCH#EYE EQU C'6NCH' IPV6 Network Configuration Header EyeCatcher.
IOC6_NCH64#EYE EQU C'6N64' IPV6 NetConfHdr EyeCatcher 64-BIT
*
* *****
* *
* * Maximum hop count for the Metric fields:
* *   GRT6RtMetric
* *   Rt6Metric
* *
* *****
*
*
IOC6_#MAXHOPMETRIC EQU 16
*
* *****
* *
* * Constants used for size of control areas
* *
* *****
*
*
IOC6_#MAXROUTES EQU 600
IOC6_#GRT6ROUTELEN EQU 44
*
* *****
* Initial buffer size for SIOCGHOMEIF6 and SIOCGRT6TABLE.
* *****
*
*
IOC6_#MAXGRT6LEN EQU 26400
IOC6_#NETCONFHDRLEN EQU 20
IOC6_#GRT6V2ROUTELEN EQU 64
IOC6_#MAXGRT6V2LEN EQU 38400
*
** BPXYIOV End

```

## BPXYIOV – Map the I/O vector structure

BPXYIOV is used by readv(), writev(), sendmsg() and recvmsg(). AMODE 31 callers use [“BPXYIOV – Map the I/O vector structure”](#) on page 1098.

```

                                SYSSTATE AMODE64=YES
                                BPXYIOV
** BPXYIOV: Socket I/O Vectors
** Used By: FCT OPN
IOV
IOV_ENTRY          DSECT ,      Array Entry
                   DS      0F
* ----- 64-bit format
IOV_BASE           DS      D      64-bit Address of buffer
IOV_LEN            DS      D      64-bit length of buffer
*
IOV#LENGTH         EQU      *-IOV_ENTRY Length of this structure
IOV_MAX            EQU      120    Maximum number of entries
** BPXYIOV End

```

## BPXYIPCQ – Map w\_getipc structure

AMODE 31 callers use [“BPXYIPCQ – Map w\\_getipc structure”](#) on page 1098.

```

                                SYSSTATE AMODE64=YES
                                BPXYIPCQ
*****
*
* BPXYIPCQ: w_getipc interface mapping
* Used By: BPXGXGET
*
*****
IPCQ               DSECT ,      Interprocess Communications - Query
IPCQLENGTH        DS      F      IPCQ#LENGTH used by system call. If not
*                               equal, check BPXYIPCQ and system levels.
IPCQTYPE          DS      CL4    "IMSG", "ISEM", "ISHM", "OVER", "IMAP"

```

```

IPCQOVER      DS      0D      OVERVIEW MAPPING STARTS HERE
*-----*
*  For IPCQTYPE = OVER, data starts here and the rest of the fields  *
*  in this section of code are not filled in.                        *
*-----*
IPCQMID       DS      FL4     MEMBER ID
IPCQKEY       DS      XL4     KEY
IPCQIPCP      DS      CL20    MAPPED BY BPXYIPCP
IPCQGTIME     DS      XL4     TIME_T OF LAST ...GET()
IPCQCTIME     DS      XL4     TIME_T OF LAST ...CTL()
IPCQTTIME     DS      XL4     TIME_T CHANGED BY TERMINATION
*-----*
*  Start of Unique data for IPCQTYPE requested                        *
*-----*
IPCQREST      DS      0C      IPCQMSG, IPCQSHM, IPCQSEM, MAPPED MEMORY
*****
* Message Queue unique data                                         *
*****
                ORG      IPCQREST
                DS      0F
IPCQBYTES     DS      F       # BYTES OF MESSAGES ON QUEUE
IPCQBBYTES    DS      F       MAX # BYTES OF MESSAGES ALLOWED ON QUEUE
IPCQLSPID     DS      F       PID OF LAST MSGSND()
IPCQLRPID     DS      F       PID OF LAST MSGRCV()
IPCQSTIME     DS      F       TIME_T OF LAST MSGSND()
IPCQRTIME     DS      F       TIME_T OF LAST MSGRCV()
IPCQNUM       DS      F       # OF MESSAGES ON QUEUE
IPCQRCNT      DS      F       COUNT OF WAITING MSGRCV
IPCQSCNT      DS      F       COUNT OF WAITING MSGSND
                DS      0CL16  MSGRCV AND MSGSND WAITERS
                DS      0CL8   MSGRCV - WAIT FOR TYPE
IPCQQRPID     DS      F       PROCESS ID
IPCQQRMSGTYPE DS      F       MESSAGE TYPE
                DS      0CL8   MSGSND - WAIT FOR ROOM TO SEND
IPCQQSPID     DS      F       PROCESS ID
IPCQQSMSGLEN  DS      F       MESSAGE LENGTH
                DS      9CL16  MSGSND AND MSGRCV WAITERS
                DS      0CL8   MESSAGES WAITING TO BE RECEIVED
IPCQQMPID     DS      F       PROCESS ID
IPCQQMMSGTYPE DS      F       MESSAGE TYPE
                DS      9CL8   MESSAGES
                DS      F       Reserved
                DS      0D
* The 64 bit time fields will be set for either 31 or 64 bit mode
* Must define storage different, depending on how assembled
* AMode 64
IPCQSTIME64   DS      FD      TIME64_T OF LAST MSGSND()
IPCQRTIME64   DS      FD      TIME64_T OF LAST MSGRCV()
IPCQQRMSGTYPE64 DS 10FD    MSGRCV 64 BIT MSG TYPE
IPCQQMMSGTYPE64 DS 10FD    MSG WAITING 64 BIT MSG TYPE
                DS      CL96   Reserved for expansion
*****
* Semaphore unique data                                           *
*****
                ORG      IPCQREST
                DS      0F
IPCQLOPID     DS      XL4     PID OF LAST SEMOP
IPCQOTIME     DS      F       TIME_T LAST SEMOP
IPCQADJBADCNT DS      F       TERMINATION BUMPS SEM_VAL LIMITS
IPCQNSEMS     DS      FL2     NUMBER OF SEMAPHORES IN THIS SET
IPCQADJCNT    DS      FL2     NUMBER OF UNDO STRUCTURES
IPCQNCNT      DS      FL2     COUNT OF WAITERS FOR >0
IPCQZCNT      DS      FL2     COUNT OF WAITERS FOR =0
                DS      0CL16  WAITERS AND ADJUSTERS
                DS      0CL8   WAITER
IPCQSWPID     DS      F       PROCESS ID
IPCQSWNUM     DS      H       SEMAPHORE NUMBER
IPCQSWOP      DS      H       SEMAPHORE OPERATION
                DS      0CL8   ADJUSTER
IPCQSAPID     DS      F       PROCESS ID
IPCQSANUM     DS      H       SEMAPHORE NUMBER
IPCQSAADJ     DS      H       SEMAPHORE OPERATION
                DS      9CL16  WAITERS AND ADJUSTERS
                DS      0D
* AMode 64
IPCQOTIME64   DS      FD      TIME64_T LAST SEMOP
                DS      CL360  Reserved for expansion
*****
* Shared Memory unique data                                       *
*****
                ORG      IPCQREST
                DS      0F

```

```

IPCQACNT      DS    F    USE COUNT (#SHMAT - #SHMDT)
               DS    F    RESERVED IN 64 BIT MODE
IPCQDTIME     DS    F    TIME_T OF LAST SHMDT()
IPCQATIME     DS    F    TIME_T OF LAST SHMAT()
IPCQLPID      DS    F    PID OF LAST SHMAT() OR SHMDT()
IPCQCPID      DS    XL4   PID OF CREATOR
*-----*
* 31 bit callers - 10 Element array of segments attached      *
*   Each element is the 4 byte PID followed by the 31 bit address *
*-----*
IPCQATPID64   DS    F    ATTACHED PROCESS ID
               DS    F    Reserved
IPCQATADDRESS64 DS AD    Segment addresses for process
               DS 18FD    Rest of elements
*
IPCQDTIME64   DS    FD    TIME_T OF LAST SHMDT()
IPCQATIME64   DS    FD    TIME_T OF LAST SHMAT()
*
IPCQSEGSZ     DS    FD    MEMORY SEGMENT SIZE
               DS    CL344 Reserved
*****
* Mapped Memory unique data                                     *
*****
               ORG    IPCQREST Mapped Memory unique data
               DS     0F
IPCQMAPCPID   DS    F    CREATOR PROCESS ID
IPCQMAPUPID   DS    F    USER PROCESS ID
IPCQMAPTOKEN  DS    2F    MAP TOKEN
IPCQMAPUID    DS    F    USER'S EFFECTIVE UID
IPCQMAPGID    DS    F    USER'S EFFECTIVE GID
IPCQMAPFLAGS  DS    XL4   FLAGS
* Flags in first byte
IPCQMAPSHUT   EQU    X'80' SHUTDOWN OF OBJECT
IPCQBLKSZ     DS    F    SIZE OF BLOCKS IN MEGS
IPCQBLKSINUSE DS    F    NUMBER OF BLOCKS IN USE
IPCQBLKSINMAP DS    F    NUMBER OF BLOCKS IN MAP AREA
IPCQBLKSMAPPED DS    F    NUMBER OF BLOCKS MAPPED
*
               DS    CL508 Reserved for expansion
*****
* Continuation of Common data                                   *
* This next ORG gets us past the largest unique section of data *
* We need to preserve the field offsets from prior releases so  *
* needed to add the rest of this common data at the end of the  *
* unique data instead of within the common area defined above.  *
*****
               ORG
IPCQGTIME64   DS    FD    TIME64_T OF LAST ...GET()
IPCQCTIME64   DS    FD    TIME64_T OF LAST ...CTL()
IPCQTTIME64   DS    FD    TIME64_T CHANGED BY TERMINATION
IPCQSECLABEL  DS    FD    SECLABEL
*****
* Overview - summary data for msgqs, semaphores, shared memory *
*****
               ORG    IPCQOVER Overview
               DS     0F    MESSAGE QUEUES
IPCQOMSGNIDS  DS    F    Maximum number MSQs allowed
IPCQOMSGHIGHH20 DS    F    Most MSQs at one time
IPCQOMSGFREE  DS    F    Number MSQs available
IPCQOMSGPRIVATE DS    F    Number MSQs with Ipc_PRIVATE
IPCQOMSGKEYED DS    F    Number MSQs with KEYS
IPCQOMSGREJECTS DS    F    TIMES MSGGET DENIED
IPCQOMSGQBYTES DS    F    MAX BYTES PER QUEUE
IPCQOMSGQNUM  DS    F    MAX NUMBER MESSAGES PER QUEUE
IPCQOMSGNOALC DS    F    # MSGSNDS THAT RETURNED ENOMEM
               DS    F
               DS     0F    SEMAPHORE
IPCQOSEMNIDS  DS    F    Maximum number SEMs allowed
IPCQOSEMHIGHH20 DS    F    Most SEMs at one time
IPCQOSEMFREE  DS    F    Number SEMs available
IPCQOSEMPRIVATE DS    F    Number SEMs with Ipc_PRIVATE
IPCQOSEMKEYED DS    F    Number SEMs with KEYS
IPCQOSEMREJECTS DS    F    TIMES SEMGET DENIED
IPCQOSEMSNSEMS DS    F    MAX NUMBER OF SEMAPHORES PER SET
IPCQOSEMSNOPS DS    F    MAX NUMBER OPERATION IN SEMOP
IPCQOSEMSBYTES DS    F    STORAGE LIMIT
IPCQOSEMCMBYTES DS    F    STORAGE COUNT
               DS    F
               DS     0F    SHARED MEMORY
IPCQOSHMNIDS  DS    F    Maximum number SHMs allowed
IPCQOSHMHIGHH20 DS    F    Most SHMs at one time
IPCQOSHMFREE  DS    F    Number SHMs available

```

```

IPCQ0SHMPRIVATE DS F Number SHMs with Ipc_PRIVATE
IPCQ0SHMKEYED DS F Number SHMs with KEYS
IPCQ0SHMREJECTS DS F TIMES SHMGET DENIED
IPCQ0SHMSPAGES DS F MAX # PAGES PER SYSTEM LIMIT
IPCQ0SHMMPAGES DS F MAX # PAGES PER SEGMENT LIMIT - ZERO
* IF 32 BITS EXCEEDED - USE
* IPCQ0SHMMPAGES64 FOR GREATER THAN 32
* BITS
IPCQ0SHMNSEGS DS F MAX # SEGMENTS PER PROCESS LIMIT
IPCQ0SHMCPAGES DS F CURRENT # BYTES SYSTEM WIDE
* This field does not include pages for
* shared memory requests processed with
* the ipc_MEGA option
IPCQ0SHMBIGGEST DS F LARGEST_SEGMENT ALLOCATED - ZERO IF
* 32 BITS EXCEEDED - USE
* IPCQ0SHMBIGGEST64 FOR GREATER THAN 32
* BITS
IPCQ0SHMMPAGES64 DS 0D MAX # PAGES PER SEGMENT LIMIT
IPCQ0SHMBIGGEST64 DS FD LARGEST_SEGMENT ALLOCATED
ORG
IPCQ#LENGTH EQU *-IPCQ Storage needed for w_getipc function
* w_getipc Command:
IPCQ#MSG EQU 1 Retrieve next message queue
IPCQ#SHM EQU 2 Retrieve next shared memory segment
IPCQ#SEM EQU 3 Retrieve next semaphore set
IPCQ#ALL EQU 4 Retrieve next member, all mechanisms
IPCQ#OVER EQU 5 Retrieve overview
IPCQ#MAP EQU 6 Retrieve mapped memory
** BPXYIPCQ End

```

## BPXYITIM — Map getitimer, setitimer structure

AMODE 31 callers use “BPXYITIM — Map getitimer, setitimer structure” on page 1101.

```

SYSSTATE AMODE64=YES
BPXYITIM
** BPXYITIM: getitimer and setitimer interval structure
** Used By: GTR STR
ITIM DSECT
** STRUCTURE OF GETITIMER (PARAMETER 2), SETITIMER (PARAMETERS 2,3)
ITIMIPAIR DS 0CL16 Initial value or value at cancel
ITIMISECONDS DS FD Seconds 0-7FFFFFFF x
DS F Padding
ITIMIMICROSEC DS 0F Microseconds 0-000F423F x
ITIMINANOSEC DS F Nanoseconds 0-369AC9FF x
ITIMRPAIR DS 0CL16 Reload Interval
ITIMRSECONDS DS FD Seconds 0-2147483647 d
DS F Padding
ITIMRMICROSEC DS 0F Microseconds 0-999999 d
ITIMRNANOSEC DS F Nanoseconds 0-999999999 d
ITIMER_REAL EQU 0 REAL TIME
ITIMER_VIRTUAL EQU 1 VIRTUAL TIME (CPU - SYSTEM)
ITIMER_PROF EQU 2 CPU TIME
ITIMER_MICRO EQU 0 1/1,000,000 of seconds
ITIMER_NANO EQU 4 1/1,000,000,000 of seconds
ITIM#LENGTH EQU 32 LENGTH THIS STRUCTURE
** BPXYITIM End

```

## BPXYMMG — Map Interface for \_map\_init and \_map\_service

AMODE 31 callers use “BPXYMMG — Map interface for \_map\_init and \_map\_service” on page 1101.

```

SYSSTATE AMODE64=YES
BPXYMMG
** BPXYMMG: BPX1MMI & BPX1MMS Interface Declares
** Used By: Callers of the BPX1MMI & BPX1MMS Interface
*
*****
* Function Code Constants
*
*****

```



```

*
MMG_INIT          EQU 1
MMG_SERVICE       EQU 2
*
*****
*
*   Parameter list mapping for the BPX1MMI MMG_INIT call
*
*****
*
_MMG_INIT_PARM    DSECT ,      MMG_INIT Parameter List
_MMG_NUMBLKS      DS    F      Fullword that contains the number of
*                               blocks to be contained in the map
*                               area.
_MMG_MEGSPERBLK   DS    F      Fullword that contains the size in
*                               megabytes of each block in the map
*                               area
_MMG_MAPTOKEN      DS    CL8    Token for map area
_MMG_AREAAADDR     DS    AD      Doubleword that contains, on input,
*                               the suggested starting address of the
*                               map area or 0. On output, this field
*                               is set to the actual map starting
*                               address.
_MMG_INIT_PARM_LEN EQU *-_MMG_INIT_PARM
*
*****
*
*   Parameter list mapping for the BPX1MMS MMG_SERVICE request
*
*   The parameter list is an array of entries, each entry having the
*   format as mapped by _MMG_SERVICE_BLK. Each entry is a request for
*   one of the supported request types: MMG_NEWBLOCK, MMG_CONN,
*   MMG_DISCONN, MMG_CNTL or MMG_FREE. In addition, an entry can be
*   marked as inactive by setting its value to MMG_NOP, which will
*   cause the entry to be skipped. The result of a given request will
*   be reflected in the array entry.
*
*   The meaning of array entry fields is dependant on the requested
*   function. The following table defines the field meanings for each
*   of the supported functions. A field not used by a service is marked
*   N/A. Fields so marked are ignored and their value is not
*   important for the specified service. All reserved fields must be
*   zero.
*
*   Function      Field      Field usage
*   -----
*   _newblock
*       _MMG_SERVICETYPE      MMG_NEWBLOCK
*       _MMG_SERVICEIFLAG     All bits should be zero except
*                               MMG_NOCONN may be set to one if
*                               the new block is to be allocated
*                               in the backing storage but not
*                               connected to the map area
*       _MMG_SERVICEOFLAG     Should be zero, but not checked
*       _MMG_Token            output
*       _MMG_BlkJAddr         input - 0 or address where the
*                               new block is to be
*                               allocated
*                               output - An address in the map
*                               area where the new
*                               block was allocated
*   _conn
*       _MMG_SERVICETYPE      MMG_CONN
*       _MMG_SERVICEIFLAG     All bits should be zero
*       _MMG_SERVICEOFLAG     Should be zero, but not checked
*       _MMG_Token            input
*       _MMG_BlkJAddr         input - 0 or address where the
*                               block identified by
*                               token is to be
*                               allocated
*                               output - An address in the map
*                               area where the block
*                               was allocated
*   _disconn
*       _MMG_SERVICETYPE      MMG_DISCONN
*       _MMG_SERVICEIFLAG     All bits should be zero except
*                               the MMG_FREE bit may be on if
*                               backing storage is to be
*                               released for the data
*       _MMG_SERVICEOFLAG     Should be zero, but not checked
*       _MMG_Token            N/A
*       _MMG_BlkJAddr         input - Address of the block

```

```

*                                     containing data to
*                                     be disconnected
*
*  _free
*      _MMG_SERVICETYPE      MMG_FREE
*      _MMG_SERVICEIFLAG     All bits should be zero
*      _MMG_SERVICEOFLAG     Should be zero, but not checked
*      _MMG_Token            input - Token of the data
*                               contained in the
*                               backing storage which
*                               is to be release
*
*      _MMG_BlKAddr          N/A
*
*  _cntl
*      _MMG_SERVICETYPE      MMG_CNTL
*      _MMG_SERVICEIFLAG     All bits should be zero except
*                               those that define the access
*                               state of the data (read or
*                               read/write flags)
*      _MMG_SERVICEOFLAG     Should be zero, but not checked
*      _MMG_Token            N/A
*      _MMG_BlKAddr          input - Address of the block
*                               containing data to be
*                               affected by the state
*                               change
*
*****
*
*  _MMG_SERVICE_PARM DSECT ,      MMG_SERVICE Parameter List
*  _MMG_SERVICE_ENTRY DS 0H
*  _MMG_SERVICETYPE DS      FL2    Type of service requested. eg, MMG_CONN
*  _MMG_SERVICEIFLAG DS      BL1    Flags
*      ORG      _MMG_SERVICEIFLAG
*  _MMG_READONLY      EQU      X'80' All pages of each area are to be made
*                               read-only
*  _MMG_READWRITE      EQU      X'40' All pages of each area are to be made
*                               read-write
*  _MMG_FREEBLOCK      EQU      X'20' The backing storage for the specified
*                               block is to be freed
*  _MMG_NOCONN          EQU      X'10' The new block is to be allocated in the
*                               backing storage but not connected to
*                               the map area
*      ORG      _MMG_SERVICEIFLAG+L'_MMG_SERVICEIFLAG
*  _MMG_SERVICEOFLAG DS      BL1    Flags
*      ORG      _MMG_SERVICEOFLAG
*  _MMG_REQFAIL          EQU      X'80' If on, a failure occurred on this entry
*                               or this entry was not processed
*      ORG      _MMG_SERVICEOFLAG+L'_MMG_SERVICEOFLAG
*  _MMG_TOKEN            DS      CL8  Token for a data block
*  _MMG_RES02B           DS      A     Reserved
*  _MMG_BLKADDR          DS      AD    Doubleword that contains the virtual
*                               address of a map area block
*  _MMG_MAXARRAYCOUNT EQU 1000      Maximum number of requests that can be
*                               in a service request array
*  _MMG_SERVICE_PARM_LEN EQU *- _MMG_SERVICE_PARM
*
*****
*
*  BPX1MMS SERVICE Request Constants (values for field
*  _MMG_SERVICETYPE)
*
*****
*
MMG_NOP                EQU 0
MMG_NEWBLOCK           EQU 1
MMG_CONN               EQU 2
MMG_DISCONN            EQU 3
MMG_FREE               EQU 4
MMG_CNTL               EQU 5
*
*****
*
** BPXYMMG End

```

## BPXYMSG — Map interprocess communication message queues

DSECT (MSGBUF) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. Default for the message size is 100 bytes. Specify VARLEN= to override this value.

AMODE 31 callers use “BPXYMSG — Map interprocess communication message queues” on page 1107.

```

SYSSTATE AMODE64=YES
BPXYMSG
** BPXYMSG: Interprocess Communication Message Queue Structure
** Used By: msgctl
MSGID_DS      DSECT ,      message queue structure
MSG_PERM      DS      CL(IPC#LENGTH) Mapped by BPXYIPCP
MSG_QNUM      DS      F      # of messages on queue
MSG_QBYTES    DS      F      max bytes allowed on queue
MSG_LSPID     DS      F      process ID of last msgsnd()
MSG_LRPID     DS      F      process ID of last msgrcv()
MSG_STIME     DS      F      time of last msgsnd()
MSG_RTIME     DS      F      time of last msgrcv()
MSG_CTIME     DS      F      time of last change get/ctl
MSG_STIME64   DS      FD     time64_t of last msgsnd()
MSG_RTIME64   DS      FD     time64_t of last msgrcv()
MSG_CTIME64   DS      FD     time64_t of last change get/ctl
MSG#LENGTH    EQU *-MSGID_DS Length of this DSECT
MSGBUF        DSECT ,      Message buffer - msgsnd, msgrcv
MSG_TYPE      DS      FD     64 bit message type
MSG_MTEXT     DS      CL100  Message text
MSGB#LENGTH   EQU *-MSGBUF Length of this DSECT
MSGXBUF       DSECT ,      Message buffer - msgxrcv
MSGX_MTIME    DS      FD     time message sent
MSGX_UID      DS      F      sender's effective UID
MSGX_GID      DS      F      sender's effective GID
MSGX_PID      DS      F      sender's PID
MSGX_TYPE     DS      FD     Message type
MSGX_MTEXT    DS      CL100  Message text
MSGX#LENGTH   EQU *-MSGXBUF Length of this DSECT
* Flag bits - msgrcv (also IPC_NOWAIT
MSG_NOERROR   EQU      4      No error if big message.
MSG_INFO      EQU      8      Use MSGXBUF not MSGBUF format
** BPXYMSG End

```

## BPXYMSGH — Map the message header

BPXYMSGH is used by the sendmsg and recvmsg syscalls. AMODE 31 callers use “BPXYMSGH — Map the message header” on page 1109.

```

SYSSTATE AMODE64=YES
BPXYMSGH
** BPXYMSGH: MSGH system call structure
** Used By: SendMsg / RecvMsg
MSGH          DSECT ,
MSGHBEGIN     DS      0D
* ----- 64-Bit Version
MSGHNAMEPTR   DS      D'0'   Pointer to sockaddr
MSGHIOVPTR    DS      D'0'   Pointer to an array of IOVEC buffers.
MSGHCONTRLPTR DS      D'0'   Pointer to ancillary data buffer
MSGHFLAGS     DS      F'0'   Output flags on received message
MSGHNAMELEN   DS      F'0'   Buffer length.
MSGHIOVNUM    DS      F'0'   Number of elements in IOVEC array.
MSGHCONTROLLEN DS      F'0'   Length of ancillary data buffer
*
* Constants
*
MSGH#LENGTH   EQU      *-MSGH Length of MsgH
*
MSGGPTR       DS      A(0)    CMsg pointer
*
MSGHDR        DSECT ,
MSGLEN        DS      F'0'   Length, including header
MSGLEVEL      DS      F'0'   Level
MSGTYPE       DS      F'0'   Type
MSGDATA       DS      0C      Data
*
* Constants
*
SCM_RIGHTS    EQU      1      Access Rights
SCM_SECINFO   EQU      16386  Security Information
*

```

```
** BPXYMSGH End
```

## BPXYOCRT – Map the OE certificate support structure

AMODE 31 callers use “BPXYOCRT – Map the OE certificate support structure” on page 1111.

```

                SYSSTATE AMODE64=YES
                BPXYOCRT
** BPXYOCRT: OE Certificate support structure
** Used By: TLS
OCRT            DSECT
OCRTTYPE        DS      F      type of certificate attached
OCRTUSERID      DS      CL9     MVS userid, null terminated, input/output
                DS      CL3     reserved
OCRTCLEN        DS      F      length of certificate associated with type
OCRTCPtr        DS      A      31-Bit ptr to the actual certificate
OCRTCERTPTR64   DS      D      64-Bit ptr to the actual certificate
OCRT_LEN        EQU      *-OCRT
OCRT_X509       EQU      1      Certificate type X509
** BPXYOCRT End

```

## BPXYPPSD – Map signal delivery data

This structure is passed to a signal interface routine (SIR). AMODE 31 callers use “BPXYPPSD – Map signal delivery data” on page 1124.

```

                SYSSTATE AMODE64=YES
                BPXYPPSD
** BPXYPPSD: Signal Data Area
** Used By: User written signal interrupt routines
PPSD            DSECT
PPSDID          DC      C'PPSD'  Eye catcher
PPSD#ID         EQU      C'PPSD'  Control Block Acronym
PPSDSP          DS      FL1      Subpool number of this PPSD
PPSD#SP         EQU      230     Subpool for the PPSD
PPSDLEN         DC      AL3(PPSD#LENGTH) Length this structure
*
* *****
* PpsdSIRParms is used to setup up a parameter list to the
* Signal Interface Routine (SIR). When the SIR is invoked, the
* address of PpsdSIRParms field is set in Register 1. The
* PpsdAddrPpsd contains the address of the Ppsd.
* *****
*
PPSDSIRPARMS    DS      0A      SIR Parameters
PPSDADDRPPSD    DC      A(PPSD) Pointer to the top of the Ppsd
PPSDSIRPARMEND  EQU      X'80'  End of Parameters flag set on
PPSDTRMEXITSTATUS DS      F      4 Byte status passed to PRTRM
PPSDSIGNUM      DS      F      Signal number
PPSDFL          DS      XL2     X'7FFF' reserved
                ORG      PPSDFL
PPSDFLAGS2A     DS      0B
PPSDQUIESCFREEZE EQU      X'80'  Interrupt due to freeze
PPSDSIRCOMPLETE EQU      X'40'  Sir done with async I/O exits
PPSDPROCDEF     EQU      X'20'  Process default
PPSDSIGQUEUE    EQU      X'10'  NSSGQ queued signal
PPSDREDRIVE     EQU      X'08'  SPB will Resend signal later
PPSDJUMPBK      EQU      X'04'  SPB return to point of interrupt
PPSDMASKONLY    EQU      X'02'  SPB restore mask only
PPSDSIGHSTOP    EQU      X'01'  Interrupt due to thread-stop
*
                ORG      PPSDFL+0001
PPSDFLAGS2B     DS      B
PPSDQUIESCEANDGET EQU      X'80'  Interrupt due to
*                                     pthread_quiesce_and_get_np
PPSDF2_64       EQU      X'40'  Use PSWxxx64 fields
PPSDACTION      DS      B      Action for this signal
*                                     catch
*                                     SIR determines default action
PPSDFLAGS       DS      B      X'00' reserved
PPSDASYNC       EQU      X'80'  Signal delivered Asynchronously

```

PPSDDUMP	EQU	X'40'	Dump for terminating signals
PPSDPTHREADKILL	EQU	X'20'	Signal sent via BPX1PTK
PPSDTHISTHREADGEN	EQU	X'10'	Sending=Receiving thread
PPSDSIGNAL	EQU	X'08'	Interrupt due to signal
PPSDCANCEL	EQU	X'04'	Interrupt due to cancel
PPSDQUIESCE	EQU	X'02'	Interrupt due to quiesce
PPSDIPT	EQU	X'01'	If ON then this is the IPT
PPSDRES1	DS	F	Reserved in 64 bit mode
PPSDSAMASK	DS	XL8	Signal mask set by BPX1SIA for this signal
*			X'00FFFFFF' reserved
PPSDSAFLAGS	DS	XL4	Do not generate SIGCHLD on stops
PPSDNOCLDSTOP	EQU	X'80'	Signal defined by signal() funct.
PPSDOLDSTYLE	EQU	X'40'	Deliver on alternate stack
PPSDONSTACK	EQU	X'20'	Reset action on delivery
PPSDRESETHAND	EQU	X'10'	Restart interruptable funcs
PPSDRESTART	EQU	X'08'	Pass sig info to catcher
PPSDSIGINF	EQU	X'04'	Don't create zombie on exit
PPSDNOCLDWAIT	EQU	X'02'	Don't block sig on delivery
PPSDNODEFER	EQU	X'01'	This is the signal mask to be set when the signal catcher returns.
PPSDCURRENTMASK	DS	XL8	Signal mask at time of interrupt except for sigsuspend case. If signal during sigsuspend, then this mask is the signal mask prior to call to sigsuspend.
*			Reserved in 64 bit mode
PPSDRES2	DS	F	Reserved in 64 bit mode
PPSDRES3	DS	F	Reserved in 64 bit mode
PPSDGENREGS	DS	CL64	Users general regs at interrupt
PPSDRES4	DS	XL8	Reserved in 64 bit mode
PPSDARREGS	DS	16F	Users AR regs at interrupt
PPSDKILDATA	DS	FL2	User specified data on BPX1KIL
PPSDKILOPTS	DS	XL2	X'7FFF' reserved
*			User specified options on BPX1KIL
PPSDPTBYPASS	EQU	X'80'	Ptrace Bypass option in effect
PPSDKERN SICODE	EQU	X'40'	PpsdKilData=Kern set SiCode
PPSDAPPLSICODE	EQU	X'20'	PpsdKilData=Appl set SiCode
PPSDCONSCANCEL	EQU	X'10'	Console MODIFY cancel qualifier
PPSDTRACEOVERRIDE	EQU	X'04'	SYSCALL Trace Override Option
PPSDTRACEACTION	EQU	X'02'	SYSCALL Trace Action Setting
*			in PpsdKilData
PPSDSUPERKILL	EQU	X'08'	Superkill option on BPX1KIL
PPSDRES5	DS	F	Reserved in 64 bit mode
PPSDLASTPTSIG	DS	F	Last Ptraced Signal
PPSDRES6	DS	2F	Reserved in 64 bit mode
PPSDSENDINGTHREAD	DS	CL8	Sending thread id
PPSDTARGETTHREAD	DS	CL8	Target thread id
PPSDSENDINGPID	DS	F	Sending process id
PPSDSENDINGUID	DS	F	Sending real uid
PPSDRES7	DS	F	Reserved in 64 bit mode
PPSDSISTATUS	DS	F	Exit status or signal
PPSDRES8	DS	F	Reserved in 64 bit mode
PPSDERRNO	DS	F	Error return code
PPSDCATCHERMASK	DS	XL8	Signal Mask to be set before signal catcher is called. If signal during sigsuspend then this field is same as mask specified on sigsuspend. If not sigsuspend, then PpsdCatcherMask and PpsdCurrentMask are equal.
*			Reserved
PPSDRES10	DS	25F	Reserved in 64 bit mode
PPSDRES9	DS	F	Time to delay signal 1000 per mic
PPSDREDRIVETIME	DS	F	Users G64H at interrupt
PPSDG64H	DS	16F	Sending thread id for MSG
PPSDRRTRMMSGTHID	DS	CL8	BPXP0101
*			Jobname of thread sending signal
PPSDSENDINGJOBNAME	DS	CL8	
*			Reserved
PPSDSAHANDLER	DS	4F	Addr of catcher function
PPSDSIR	DS	AD	Addr Signal interrupt routine
PPSDUSERDATA	DS	AD	User data specified on BPX1MSS
PPSDPSW	DS	XL16	Users PSW at interrupt
PPSDQUIESCEDATA	DS	FD	Quiesce_Data specified on BPX1QUT
*			User_Data specified on BPX1SIA
PPSDSIGACTIONDATA	DS	FD	Threads workarea address specified on BPX1PTC (pthread_create). This address is zero if the thread was not pt_created.
PPSDPTXLWAPTR	DS	AD	Address of faulting instruction for SIGILL, SIGFPE, SIGSEGV
*			
PPSDSIADDR	DS	AD	
*			

PPSDSIBAND	DS	FD	Band event
PPSDSQV	DS	FD	Signal si_value
	DS	4F	Reserved
	DS	FL2	Reserved
PPSDAIOCB64	DS	FL2	Amode(64) Exit Flags
PPSDEXCOUNT	DS	FL2	Count of PpsdAioCb's
PPSDEXLASTIX	DS	FL2	Last array index used
PPSDAIOCB	DS	12D	AioCb Array for Async Exit
PPSDEND	DS	0D	End of PPSD on double word
PPSD#LENGTH	EQU	*-PPSD	Length of this structure
** BPXYPPSD End			

## BPXYPTXL – Map the parameter pist for pthread\_create

AMODE 31 callers use “BPXYPTXL – Map the parameter list for pthread\_create” on page 1141.

```

SYSSTATE AMODE64=YES
BPXYPTXL
** BPXYPTXL: Pthread Parameter List
** Used By: PTX
PTXL
PTXLWORKAREAPTR DSECT , Parm List returned by BPX1PTX
PTXLATTRIBUTEPTR DS AD Pointer to User Work Area
PTXLTHIDPTR DS AD Pointer to User Attributes
PTXLTHIDPTR DS A Pointer to Thread ID
PTXLSTATUSPTR DS A Pointer to Thread Run Status
PTXL#LENGTH EQU *-PTXL
PTXLRs DSECT , Thread Run Status
PTXLRs DS 0F
PTXLRsFLAGS DS 0BL4 Thread Run Status Flags
PTXLRsFLAGS0 DS B 1st byte
PTXLRsREADY EQU X'80' Thread is ready to run
PTXLRsFLAGS1 DS B 2nd byte
PTXLRsFLAGS2 DS B 3rd byte
PTXLRsFLAGS3 DS B 4th byte
PTXLRs#LENGTH EQU *-PTXLRs
** BPXYPTXL End

```

## BPXYRLIM – Map the rlimit, rusage, and timeval structures

AMODE 31 callers use “BPXYRLIM – Map the rlimit, rusage, and timeval structures” on page 1142.

```

SYSSTATE AMODE64=YES
BPXYRLIM
** BPXYRLIM: Rlimit, Timeval, and Rusage Structures
** Used By: setrlimit, getrlimit, and getrusage
RLIMIT DSECT , Rlimit structure
RLIM_CUR_DW DS 0CL8 Current limit (doubleword)
RLIM_CUR_HW DS F Current (soft) limit highword - X
used only for RLIMIT_FSIZE X
and RLIMIT_MEMLIMIT, it is X
ignored for all other resources
RLIM_CUR DS 0F Current (soft) limit lowword
RLIM_CUR_LW DS F Current (soft) limit lowword
RLIM_MAX_DW DS 0CL8 Current limit (doubleword)
RLIM_MAX_HW DS F Current (hard) limit highword - X
used only for RLIMIT_FSIZE X
and RLIMIT_MEMLIMIT, it is X
ignored for all other resources
RLIM_MAX DS 0F Maximum (hard) limit lowword
RLIM_MAX_LW DS F Maximum (hard) limit lowword
RLIMIT#LENGTH EQU *-RLIMIT Length of this DSECT
TIMEVAL DSECT , Timeval structure
TMVL_SEC DS FD Seconds
TMVL_PADDING DS F Padding
TMVL_USEC DS F Microseconds
TIMEVAL#LENGTH EQU *-TIMEVAL Length of this DSECT
RUSAGE DSECT , Rusage structure
RU_UTIME DS CL(TIMEVAL#LENGTH) User time used
RU_STIME DS CL(TIMEVAL#LENGTH) System time used
RUSAGE#LENGTH EQU *-RUSAGE Length of this DSECT

```

```
** BPXYRLIM End
```

## BPXYSELT — Map the timeout value for the select syscall

AMODE 31 callers use “BPXYSELT — Map the timeout value for the select syscall ” on page 1145.

```

SYSSTATE AMODE64=YES
BPXYSELT ,
** BPXYSELT: Select Time Structure
** Used By: Select Syscall
SELT          DSECT ,
SELTBEGIN     DS      0D
*-----64-bit format
*
TV_SEC        DS      D'0'   Seconds
              DS      F'0'   Padding
TV_USEC       DS      F'0'   Microseconds
* Constants
*
SELT#LENGTH   EQU      *-SELT Length of SELT
** BPXYSELT End

```

## BPXYSEM — Map interprocess communication semaphores

DSECTs (SEMID\_DS, SEM\_ARRAY and SEM\_BUF\_ELE) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. AMODE 31 callers use “BPXYSEM — Map interprocess communication semaphores” on page 1146.

```

SYSSTATE AMODE64=YES
BPXYSEM ,
** BPXYSEM: Interprocess Communications Permission
** Used By: XS0, XSC
SEMID_DS      DSECT ,      semctl structure
SEM_PERM      DS      CL(IPC#LENGTH) Mapped by BPXYIPCP
SEM_NSEMS     DS      H      number of semaphores in set
              DS      H      spacer
SEM_OTIME     DS      FL4    last semop() time
SEM_CTIME     DS      FL4    last time changed by semctl()
SEM_OTIME64   DS      FD      last semop() time64_t
SEM_CTIME64   DS      FD      last semctl() time64_t
SEM#LENGTH    EQU      *-SEMID_DS Length of this DSECT
* SETVAL - a one element array for Semaphore_Number
* SETALL, GETALL - an array with Number_of Semaphore elements
SEM_ARRAY     DSECT ,      SETALL, GETALL, SETVAL
SEM_ARRAY_VAL DS      FL2    semaphore value
SEM_BUF_ELE   DSECT ,      sembuf element - semop
SEM_NUM       DS      FL2    semaphore number (0 to n-1)
SEM_OP        DS      FL2    semaphore operation
SEM_FLG       DS      H      operation flags
SEM#BUFLLEN   EQU      *-SEM_BUF_ELE
* Flag bits - semop (also IPC_NOWAIT)
SEM_UNDO      EQU      2      Set up adjust on exit entry.
* Control Commands - (also IPC_RMID, IPC_SET, IPC_STAT):
SEM_GETVAL    EQU      21     Get the current semaphore value
SEM_SETVAL    EQU      22     Change the semaphore value
SEM_GETPID    EQU      23     Get PID of last process to alter sem
SEM_GETNCNT   EQU      24     Get count of tasks waiting for val>0
SEM_GETZCNT   EQU      25     Get count of tasks waiting for val=0
SEM_GETALL    EQU      26     Get the current semaphore values
SEM_SETALL    EQU      27     Change the semaphore values
* Maximum and minimum values
SEM#MAX_VAL   EQU      32767  Maximum sem_val (min = 0)
SEM#MAX_ADJ   EQU      16383  Maximum sem_adj (min = -MAX)
** BPXYSEM End

```

## BPXYSFPL — Map the send\_file parameter list

AMODE 31 callers use “BPXYSFPL — Map the send\_file parameter list” on page 1147.

```

                SYSSTATE AMODE64=YES
                BPXYSFPL      ,
** BPXYSFPL: SFPL system call structure
** Used By: BPX1SF
SFPL           DSECT      ,
SFsocketDES    DS         F      Socket Descriptor
SFheaderLEN    DS         F      Header Length
SFheaderVPTR   DS         0F
SFheaderALET   DS         F      Header Alet
                DS         F      64-bit pointer below
SFFILEDES      DS         F      File Descriptor
SFFILEBYTESDW  DS         0F     Bytes to send Double Word (-1=all)
SFFILEBYTESH   DS         F      High Word
SFFILEBYTESL   DS         F      Low Word
SFFILEOFFSETDW DS         0F     Offset Double Word
SFFILEOFFSETH  DS         F      High Word
SFFILEOFFSETL  DS         F      Low Word
SFFILESIZEWD   DS         0F     File Size Double Word
SFFILESIZEH   DS         F      High Word
SFFILESIZEL   DS         F      Low Word
SFTRAILERLEN   DS         F      Trailer Length
SFTRAILERVPTR  DS         0F
SFTRAILERALET  DS         F      Trailer Alet
                DS         F      64-bit pointer below
SFBYTESENTDW   DS         0F     Bytes Sent Double Word
SFBYTESENTH    DS         F      High Word
SFBYTESENTL    DS         F      Low Word
SFFLAGS        DS         0XL4   Control Flags
SFPLVERSION    DS         XL1    Version
SFFLAGBYTE2    DS         XL1    Reserved
SFFLAGBYTE3    DS         XL1    Reserved
SFFLAGBYTE4    DS         XL1    Flags
SF_CLOSE       EQU        2      Close Socket Descriptor
SF_REUSE       EQU        1      Reuse Socket Descriptor
SFHEADERPTR    DS         FL8    Header Ptr
SFTRAILERPTR   DS         FL8    Trailer Ptr
SFRESERVE      DS         CL12   Reserved
*
SFPLEND        EQU        *
*
SFPL#LENGTH    EQU        SFPLEND-SFPL
*
*   Constants
*
** BPXYSFPL End

```

## BPXYSHM—Map interprocess communication shared memory segments

AMODE 31 callers use “BPXYSHM—Map interprocess communication shared memory segments” on page 1147.

```

                SYSSTATE AMODE64=YES
                BPXYSHM      ,
** BPXYSHM: Interprocess Communications Permission
** Used By: XMC
SHMID_DS       DSECT      ,      SHMID_DS - shmctl structure
SHM_PERM       DS         CL(IPC#LENGTH) Mapped by BPXYIPC
SHM_SEGSZ      DS         F      size of segment in bytes
SHM_LPID       DS         F      process ID of last operation
SHM_CPID       DS         F      process ID of creator
SHM_NATTCH     DS         F      number of current attaches
SHM_ETIME      DS         F      time of last shmat
SHM_DTIME      DS         F      time of last shmdt
SHM_CTIME      DS         F      time of last change shmget/shmctl
SHM_RES2       DS         F      Reserved
SHM_FLAGS      DS         F      Flags

```



```

SHM_SEG64      EQU  X'80'  Shared memory above the bar
                ORG  SHM_FLAGS+1
SHM_DUMP_PRI064 DS  FL1    Dump priority for this seg
SHM_RES3       DS  FL2    Reserved
SHM_SEGADDR64  DS  AD      Address of segment
SHM_SEGSIZE64  DS  FD      Size of segment in bytes
SHM_ETIME64    DS  FD      time64_t of last shmat
SHM_DTIME64    DS  FD      time64_t of last shmdt
SHM_CTIME64    DS  FD      time64_t of last change shmget/shmctl
*
* Mode bits (mapped over S_TYPE in BPXYMODE):
SHM_RDONLY     EQU  1      Attach read-only (else read-write)
SHM_RND        EQU  2      Round attach address to SHMLBA
SHMLBA         EQU  4096   Rounding boundary
SHM#LENGTH     EQU  *-SHMID_DS  Length of this DSECT
** BPXYSHM End

```

## BPXYSINF — Map SIGINFO\_T structure

DSECT (SIGINFO\_T) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. AMODE 31 callers use “BPXYSINF — Map SIGINFO\_T structure” on page 1150.

```

                SYSSTATE AMODE64=YES
                BPXYSINF
** BPXYSINF: siginfo_t Structure
** Used By: waitid
SIGINFO_T      DSECT ,      Siginfo_t structure
SI_SIGNO       DS  F        signal number
SI_ERRNO       DS  F        error number
SI_CODE        DS  F        signal code
SI_PID         DS  F        sending process ID
SI_UID         DS  F        real user ID of sending process
SI_RES01       DS  F        reserved in 64 bit mode
SI_STATUS      DS  F        exit value or signal
SI_RES02       DS  F        reserved in 64 bit mode
SI_RES03       DS  F        reserved in 64 bit mode
SI_RES04       DS  F        reserved in 64 bit mode
SI_ADDR        DS  AD       address of faulting instruction
SI_BAND        DS  FD       band event for SIGPOLL
SI_VALUE       DS  FD       signal value
SIGINFO#LENGTH EQU  *-SIGINFO_T Length of this DSECT
** BPXYSINF End

```

## BPXYSSET — Map the sigaction set

DSECT=.. is not supported. The generated code will allocate SSETOPTION\_FLAGS and a DSECT for SSET. This should be followed by CSECT statement to return to the current DSECT or CSECT. AMODE 31 callers use “BPXYSSET — Map the sigaction set” on page 1160.

```

                SYSSTATE AMODE64=YES
                BPXYSSET
** BPXYSSET: Macro which enables multiple signal calls
** Used By: SA2
SSETOPTION_FLAGS DS  0F
SSETOPTION_FLAGS1 DS  FL1  FLAGS INDICATING CALLER OPTIONS
SSET_IGINVALID   EQU  X'80' IGNORE INVALID SIGNALS & SIGACTIONS      X
                                0=DO NOT IGNORE, 1=IGNORE
                                DS  3FL1  RESERVED
SSET             DSECT ,
SSETCONSOLMASK   DS  XL8  SIGNALS HAVING THE SAME FLAGS,MASK,      X
                                USERDATA, AND SIGNAL ACTION
SSETCOMPARE      DS  0CL28
SSETSAHANDLER    DS  AD    ADDRESS OF A SIGNAL HANDLER ROUTINE
SSETSAMASK       DS  XL8  VALUE FOR SIGACTION MASK
SSETUSERDATA     DS  FD    USER DEFINED DATA
SSETFLAGS        DS  XL4  VALUE FOR SIGACTION FLAGS (BPXYSIGH)
SSETRES01        DS  F     Reserved
SSET#LENGTH      EQU  *-SSET  LENGTH OF ONE SSET ENTRY

```

```
** BPXYSSET End
```

## BPXYWLM — WLM constants and parameter list DSECTs

BPXYWLM work load manager constants and DSECTs. AMODE 31 callers use [“BPXYWLM — WLM constants and parameter list DSECTs”](#) on page 1173.

```

        SYSSTATE AMODE64=YES
        BPXYWLM
** BPXYWLM: BPX1WLM Interface Declares
** Used By: Callers of the BPX1WLM Interface
*
*   BPX1WLM Function Code Constants
*
WLM_QUERY_METRICS      EQU 1
WLM_QUERY_SCHEDENV     EQU 2
WLM_CHECK_SCHEDENV     EQU 3
WLM_DISCONNECT         EQU 4
WLM_DELETE_WORKUNIT    EQU 5
WLM_JOIN_WORKUNIT      EQU 6
WLM_LEAVE_WORKUNIT     EQU 7
WLM_CONNECT_WORKMGR    EQU 8
WLM_CONNECT_SERVERMGR  EQU 9
WLM_CREATE_WORKUNIT    EQU 10
WLM_CONTINUE_WORKUNIT  EQU 11
WLM_EXTRACT_WORKUNIT   EQU 12
WLM_EXPORT_WORKUNIT    EQU 13
WLM_UNDOEXPORT_WORKUNIT EQU 14
WLM_IMPORT_WORKUNIT    EQU 15
WLM_UNDOIMPORT_WORKUNIT EQU 16
WLM_QUERY_ENCLAVECLASS EQU 17
WLM_CONNECT_EXPORTIMPORT EQU 18
*   Function codes 100-112 are reserved
ARM_BIND_THREAD        EQU 200
ARM_BLOCK_TRANSACTION  EQU 201
ARM_DESTROY_APPLICATION EQU 202
ARM_DISCARD_TRANSACTION EQU 203
ARM_GENERATE_CORRELATOR EQU 204
ARM_GET_ARRIVAL_TIME   EQU 205
ARM_REGISTER_APPLICATION EQU 206
ARM_REGISTER_METRIC     EQU 207
ARM_REGISTER_TRANSACTION EQU 208
ARM_REPORT_TRANSACTION  EQU 209
ARM_START_APPLICATION  EQU 210
ARM_START_TRANSACTION  EQU 211
ARM_STOP_APPLICATION   EQU 212
ARM_STOP_TRANSACTION   EQU 213
ARM_UNBIND_THREAD      EQU 214
ARM_UNBLOCK_TRANSACTION EQU 215
ARM_UPDATE_TRANSACTION  EQU 216
EWLM_CLASSIFY_CORRELATOR EQU 217
*   BPX1WLM/BPX4WLM Parameter List Mappings
*
_WQM          DSECT ,      WLM_QUERY_METRICS Parameter List
_WQM_SYSI_PTR DS    AD      Address of a fullword pointer that
*                          contains the address of the buffer
*                          to return the WLM system information.
*                          This data is returned in the format
*                          of the IWMWSYSI mapping macro.
_WQM_SYSI_LEN DS    AD      Address of a fullword that contains
*                          the length of the buffer to return
*                          the WLM system information
_WQM_END      DS    0C      End of WQM
*
_WQS          DSECT ,      WLM_QUERY_SCHEDENV Parameter List
_WQS_SETH_PTR DS    AD      Address of a fullword pointer that
*                          contains the address of the buffer
*                          to return the WLM scheduling
*                          environment information.
*                          This data is returned in the format
*                          of the IWMSET mapping macro.
_WQS_SETH_LEN DS    AD      Address of a fullword that contains
*                          the length of the buffer to return
*                          the WLM scheduling environment data.
_WQS_END      DS    0C      End of _WQS

```

* _WCS	DSECT ,	WLM_CHECK_SCHEDENV Parameter List
_WCS_SCH_ENV	DS AD	Address of a 16 byte character string
* *		that contains the scheduling
* _WCS_SYS_NAME	DS AD	environment to be checked.
* *		Address of a 8 byte character string
* _WCS_END	DS 0C	that contains the system name to be
* *		checked.
* _WDC	DSECT ,	End of _WCS
_WDC_CONN_TKN	DS AD	WLM_DISCONNECT Parameter List
* *		Address of an fullword that contains
* _WDC_END	DS 0C	the connect token to be disconnected
* *		from.
* _WDW	DSECT ,	End of _WDC
_WDW_ENC_TKN	DS AD	WLM_DELETE_WORKUNIT Parameter List
* *		Address of a doubleword that contains
* _WDW_END	DS 0C	the WLM enclave token representing the
* *		work unit to be deleted.
* _WJW	DSECT ,	End of _WDW
_WJW_ENC_TKN	DS AD	WLM_JOIN_WORKUNIT Parameter List
* *		Address of a doubleword that contains
* _WJW_END	DS 0C	the WLM enclave token representing the
* *		work unit to join.
* _WLW	DSECT ,	End of _WJW
_WLW_ENC_TKN	DS AD	WLM_LEAVE_WORKUNIT Parameter List
* *		Address of a doubleword that contains
* _WLW_END	DS 0C	the WLM enclave token representing the
* *		work unit to leave.
* _WNW	DSECT ,	End of _WLW
_WNW_ENC_TKN	DS AD	WLM_CONTINUE_WORKUNIT Parameter List
* *		Address of a doubleword to return the
* _WNW_END	DS 0C	the WLM enclave token of the created
* *		work unit.
* _WCW	DSECT ,	End of _WNW
_WCW_ENC_TKN	DS AD	WLM_CREATE_WORKUNIT Parameter List
* *		Address of a doubleword to return the
* _WCW_CLASSIFY	DS AD	the WLM enclave token of the created
* *		work unit.
* _WCW_ARR_TIME	DS AD	Address of a fullword pointer that
* *		contains the address of a IWMCLSFY
* _WCW_FUNC_NAME	DS AD	Parameter List.
* *		Address of a doubleword field that
* _WCW_END	DS 0C	contains the arrival time of the
* *		work request in STCK format.
* _WSC	DSECT ,	Address of a 8 byte character string
_WSC_SUB_SYS	DS AD	that contains the descriptive function
* *		name of the work request.
* _WSC_SUB_SYS_NM	DS AD	End of _WCW
* *		WLM_CONNECT_SERVERMGR Parameter List
* _WSC_APPL_ENV	DS AD	Address of a 4 byte character string
* *		that contains the subsystem type the
* _WSC_PAR_EU	DS AD	server manager is requesting connection
* *		for.
* _WSC_END	DS 0C	Address of a 8 byte character string
* *		that contains the subsystem name the
* _WWC	DSECT ,	server manager is requesting connection
_WWC_SUB_SYS	DS AD	for.
* *		Address of a 32 byte character string
* *		that contains the application
* *		environment name associated with the
* *		server.
* *		Address of a fullword that contains
* *		number of parallel execution units
* *		in the server environment.
* *		End of _WSC
* *		WLM_CONNECT_WORKMGR Parameter List
* *		Address of a 4 byte character string
* *		that contains the subsystem type the
* *		work manager is requesting connection

* _WWC_SUB_SYS_NM	DS	AD	for.
* * * * _WWC_END	DS	0C	Address of a 8 byte character string that contains the subsystem name the work manager is requesting connection for. End of _WWC
* _WEW	DSECT	,	WLM_EXTRACT_WORKUNIT Parameter List
* _WEW_ENC_TKN	DS	AD	Address of a doubleword that contains the WLM enclave token representing the active work unit.
* _WEW_END	DS	0C	End of _WEW
* _WXW	DSECT	,	WLM_EXPORT_WORKUNIT Parameter List
* _WXW_ENC_TKN	DS	AD	Address of a doubleword that contains the WLM enclave token representing the work unit to be exported.
* _WXW_EXP_TKN	DS	AD	Address of the 32 bytes to return the WLM export token of the exported work unit.
* _WXW_CONN_TKN	DS	AD	Address of a fullword that contains the connect token associated with the workmanager.
* _WXW_END	DS	0C	End of _WXW
* _WUXW	DSECT	,	WLM_UNEXPORT_WORKUNIT Parameter List
* _WUXW_EXP_TKN	DS	AD	Address of the 32 bytes that contains the WLM export token representing the exported work unit.
* _WUXW_CONN_TKN	DS	AD	Address of a fullword that contains the connect token associated with the workmanager.
* _WUXW_END	DS	0C	End of _WUXW
* _WIW	DSECT	,	WLM_IMPORT_WORKUNIT Parameter List
* _WIW_EXP_TKN	DS	AD	Address of the 32 bytes that contains the WLM export token representing the exported work unit.
* _WIW_ENC_TKN	DS	AD	Address of a doubleword to return the WLM enclave token of the imported work unit.
* _WIW_CONN_TKN	DS	AD	Address of a fullword that contains the connect token associated with the workmanager.
* _WIW_END	DS	0C	End of _WIW
* _WUIW	DSECT	,	WLM_UNIMPORT_WORKUNIT Parameter List
* _WUIW_EXP_TKN	DS	AD	Address of the 32 bytes that contains the WLM export token representing the imported work unit.
* _WUIW_CONN_TKN	DS	AD	Address of a fullword that contains the connect token associated with the workmanager.
* _WUIW_END	DS	0C	End of _WUIW
* _WQEC	DSECT	,	WLM_QUERY_ENCLAVECLASS Parameter List
* _WQEC_ENC_TKN	DS	AD	Address of a doubleword that contains the WLM enclave token representing the work unit to be queried.
* _WQEC_SYSEC_PTR	DS	AD	Address of a fullword pointer that contains the address of the buffer to return the WLM Query Enclave Data.
* * * _WQEC_SYSEC_LEN	DS	AD	This data is returned in the format of the IWMECD mapping macro. Address of a fullword that contains the length of the buffer to return the WLM Query Enclave Data.
* _WQEC_END	DS	0C	End of WQEC
* _WCEI	DSECT	,	WLM_CONNECT_EXPORTIMPORT Parameter List
* _WCEI_SUB_SYS	DS	AD	Address of a 4 byte character string that contains the subsystem type the work manager is requesting connection for.
* _WCEI_SUB_SYS_NM	DS	AD	Address of a 8 byte character string

*			that contains the subsystem name the
*			work manager is requesting connection
*			for.
_WCEI_END	DS	0C	End of _WCEI
*			
_ABI	DSECT	,	ARM_BIND_THREAD Parameter
*			List
_ABI_CONTEXT	DS	AD	Reserved.
*			Must be zero.
_ABI_TRAN_HDL	DS	AD	Address of a 8 byte field that
*			contains the transaction handle.
_ABI_FLAGS	DS	AD	Address of a 4 byte field
*			that contains flags.
_ABI_BUFFER4	DS	AD	Address of a data area that
*			contains additional input data.
_ABI_END	DS	0C	End of _ABI
*			
_ABT	DSECT	,	ARM_BLOCK_TRANSACTION Parameter
*			List
_ABT_CONTEXT	DS	AD	Reserved.
*			Must be zero.
_ABT_TRAN_HDL	DS	AD	Address of a 8 byte field that
*			contains the transaction handle.
_ABT_FLAGS	DS	AD	Address of a 4 byte field
*			that contains flags.
_ABT_BUFFER4	DS	AD	Address of a data area that
*			contains additional input data.
_ABT_BLOCK_HDL	DS	AD	Address of a fullword pointer that
*			contains the address of the 8 byte
*			field to return the block handle.
_ABT_END	DS	0C	End of _ABT
*			
_ADA	DSECT	,	ARM_DESTROY_APPLICATION Parameter
*			List
_ADA_CONTEXT	DS	AD	Reserved.
*			Must be zero.
_ADA_APPL_ID	DS	AD	Address of a 16 byte field
*			that contains the application ID.
_ADA_FLAGS	DS	AD	Address of a 4 byte field
*			that contains flags.
_ADA_BUFFER4	DS	AD	Address of a data area that
*			contains additional input data.
_ADA_END	DS	0C	End of _ADA
*			
_ADT	DSECT	,	ARM_DISCARD_TRANSACTION Parameter
*			List
_ADT_CONTEXT	DS	AD	Reserved.
*			Must be zero.
_ADT_TRAN_HDL	DS	AD	Address of a 8 byte field that
*			contains the transaction handle.
_ADT_FLAGS	DS	AD	Address of a 4 byte field
*			that contains flags.
_ADT_BUFFER4	DS	AD	Address of a data area that
*			contains additional input data.
_ADT_END	DS	0C	End of _ADT
*			
_AGC	DSECT	,	ARM_GENERATE_CORRELATOR Parameter
*			List
_AGC_CONTEXT	DS	AD	Reserved.
*			Must be zero.
_AGC_APP_HDL	DS	AD	Address of a 8 byte field that
*			contains the application handle.
_AGC_TRAN_ID	DS	AD	Address of a 16 byte field that
*			contains the transaction ID.
_AGC_PAR_CORR	DS	AD	Address of a data area that
*			contains the parent correlator.
_AGC_FLAGS	DS	AD	Address of a 4 byte field
*			that contains flags.
_AGC_BUFFER4	DS	AD	Address of a data area that
*			contains additional input data.
_AGC_CUR_CORR	DS	AD	Address of a fullword pointer that
*			contains the address of the buffer
*			to return the current correlator.
_AGC_END	DS	0C	End of _AGC
*			
_AGT	DSECT	,	ARM_GET_ARRIVAL_TIME Parameter
*			List
_AGT_CONTEXT	DS	AD	Reserved.
*			Must be zero.
_AGT_TIMESTAMP	DS	AD	Address of a fullword pointer that
*			contains the address of a 64 bit
*			field to return the arrival time.

_AGT_END	DS	0C	End of _AGT
*_ARA	DSECT	,	ARM_REGISTER_APPLICATION Parameter List
*_ARA_CONTEXT	DS	AD	Reserved.
*_ARA_APP_NAME	DS	AD	Must be zero.
*_ARA_IN_APP_ID	DS	AD	Address of a character string that contains the application name.
*_ARA_FLAGS	DS	AD	Address of a 16 byte field that contains an input application ID.
*_ARA_BUFFER4	DS	AD	Address of a 4 byte field that contains flags.
*_ARA_OUT_APP_ID	DS	AD	Address of a data area that contains additional input data.
*_ARA_END	DS	0C	Address of a fullword pointer that contains the address of a 16 byte field to return the output application ID.
*_AMR	DSECT	,	End of _ARA
*_AMR_CONTEXT	DS	AD	ARM_REGISTER_METRIC Parameter List
*_AMR_APP_ID	DS	AD	Reserved.
*_AMR_MET_NAME	DS	AD	Must be zero.
*_AMR_MET_FORMAT	DS	AD	Address of a 16 byte field that contains the application ID.
*_AMR_MET_USAGE	DS	AD	Address of a character string that contains the metric name.
*_AMR_UNIT	DS	AD	Address of a 1 byte field that contains the metric format.
*_AMR_IN_MET_ID	DS	AD	Address of a 2 byte field that contains the metric usage.
*_AMR_FLAGS	DS	AD	Address of a character string that contains the units of the metric.
*_AMR_BUFFER4	DS	AD	Address of a 16 byte field that contains an input metric ID.
*_AMR_OUT_MET_ID	DS	AD	Address of a 4 byte field that contains flags.
*_AMR_END	DS	0C	Address of a data area that contains additional input data.
*_ART	DSECT	,	Address of a fullword pointer that contains the address of a 16 byte field to return the output metric ID.
*_ART_CONTEXT	DS	AD	End of _AMR
*_ART_APP_ID	DS	AD	ARM_REGISTER_TRANSACTION Parameter List
*_ART_TRAN_NAME	DS	AD	Reserved.
*_ART_IN_TRAN_ID	DS	AD	Must be zero.
*_ART_FLAGS	DS	AD	Address of a 16 byte field that contains the application ID.
*_ART_BUFFER4	DS	AD	Address of a character string that contains the transaction name.
*_ART_OUT_TRAN_ID	DS	AD	Address of a 16 byte field that contains an input transaction ID.
*_ART_END	DS	0C	Address of a 4 byte field that contains flags.
*_ATR	DSECT	,	Address of a data area that contains additional input data.
*_ATR_CONTEXT	DS	AD	Address of a fullword pointer that contains the address of a 16 byte field to return the output transaction ID.
*_ATR_APP_HDL	DS	AD	End of _ART
*_ATR_TRAN_ID	DS	AD	ARM_REPORT_TRANSACTION Parameter List
*_ATR_TRAN_STA	DS	AD	Reserved.
*_ATR_RESP_TIME	DS	AD	Must be zero.
*_ATR_STOP_TIME	DS	AD	Address of a 8 byte field that contains the application handle.
*_ATR_PAR_CORR	DS	AD	Address of a 16 byte field that contains the transaction ID.
*			Address of a 4 byte field that contains the transaction status.
			Address of a 64 bit field that contains the response time.
			Address of a 64 bit field that contains the stop time.
			Address of a data area that contains the parent correlator.

_ATR_CUR_CORR	DS	AD	Address of a data area that contains the current correlator.
*_ATR_FLAGS	DS	AD	Address of a 4 byte field that contains flags.
*_ATR_BUFFER4	DS	AD	Address of a data area that contains additional input data.
*_ATR_END	DS	0C	End of _ATR
*_AAS	DSECT	,	ARM_START_APPLICATION Parameter List
*_AAS_CONTEXT	DS	AD	Reserved. Must be zero.
*_AAS_APP_ID	DS	AD	Address of a 16 byte field that contains the application ID.
*_AAS_APP_GRP	DS	AD	Address of a character string that contains the application group name.
*_AAS_APP_INS	DS	AD	Address of a character string that contains the application instance name.
*_AAS_FLAGS	DS	AD	Address of a 4 byte field that contains flags.
*_AAS_BUFFER4	DS	AD	Address of a data area that contains additional input data.
*_AAS_APP_HDL	DS	AD	Address of a fullword pointer that contains the address of the 8 byte field to return the application handle.
*_AAS_END	DS	0C	End of _AAS
*_AST	DSECT	,	ARM_START_TRANSACTION Parameter List
*_AST_CONTEXT	DS	AD	Reserved. Must be zero.
*_AST_APP_HDL	DS	AD	Address of a 8 byte field that contains the application handle.
*_AST_TRAN_ID	DS	AD	Address of a 16 byte field that contains the transaction ID.
*_AST_PAR_CORR	DS	AD	Address of a data area that contains the parent correlator.
*_AST_FLAGS	DS	AD	Address of a 4 byte field that contains flags.
*_AST_BUFFER4	DS	AD	Address of a data area that contains additional input data.
*_AST_TRAN_HDL	DS	AD	Address of a fullword pointer that contains the address of the 8 byte field to return the transaction handle.
*_AST_CUR_CORR	DS	AD	Address of a fullword pointer that contains the address of the buffer to return the current correlator.
*_AST_END	DS	0C	End of _AST
*_APA	DSECT	,	ARM_STOP_APPLICATION Parameter List
*_APA_CONTEXT	DS	AD	Reserved. Must be zero.
*_APA_APP_HDL	DS	AD	Address of a 8 byte field that contains the application handle.
*_APA_FLAGS	DS	AD	Address of a 4 byte field that contains flags.
*_APA_BUFFER4	DS	AD	Address of a data area that contains additional input data.
*_APA_END	DS	0C	End of _APA
*_APT	DSECT	,	ARM_STOP_TRANSACTION Parameter List
*_APT_CONTEXT	DS	AD	Reserved. Must be zero.
*_APT_TRAN_HDL	DS	AD	Address of a 8 byte field that contains the transaction handle.
*_APT_TRAN_STA	DS	AD	Address of a 4 byte number that contains the transaction status.
*_APT_FLAGS	DS	AD	Address of a 4 byte field that contains flags.
*_APT_BUFFER4	DS	AD	Address of a data area that contains additional input data.
*_APT_END	DS	0C	End of _APT
*_AUB	DSECT	,	ARM_UNBIND_THREAD Parameter List
*_AUB_CONTEXT	DS	AD	Reserved.

*				Must be zero.
_AUB_TRAN_HDL	DS	AD		Address of a 8 byte field that contains the transaction handle.
*				
_AUB_FLAGS	DS	AD		Address of a 4 byte field that contains flags.
*				
_AUB_BUFFER4	DS	AD		Address of a data area that contains additional input data.
*				
_AUB_END	DS	0C		End of _AUB
*				
_AUT	DSECT	,		ARM_UNBLOCK_TRANSACTION Parameter List
*				
_AUT_CONTEXT	DS	AD		Reserved.
*				Must be zero.
_AUT_TRAN_HDL	DS	AD		Address of a 8 byte field that contains the transaction handle.
*				
_AUT_BLOCK_HDL	DS	AD		Address of a 8 byte field that contains the block handle.
*				
_AUT_FLAGS	DS	AD		Address of a 4 byte field that contains flags.
*				
_AUT_BUFFER4	DS	AD		Address of a data area that contains additional input data.
*				
_AUT_END	DS	0C		End of _AUT
*				
_AUP	DSECT	,		ARM_UPDATE_TRANSACTION Parameter List
*				
_AUP_CONTEXT	DS	AD		Reserved.
*				Must be zero.
_AUP_TRAN_HDL	DS	AD		Address of a 8 byte field that contains the transaction handle.
*				
_AUP_FLAGS	DS	AD		Address of a 4 byte field that contains flags.
*				
_AUP_BUFFER4	DS	AD		Address of a data area that contains additional input data.
*				
_AUP_END	DS	0C		End of _AUP
*				
_ACC	DSECT	,		EWLM_CLASSIFY_CORRELATOR Parameter List
*				
_ACC_CONTEXT	DS	AD		Reserved.
*				Must be zero.
_ACC_APP_HDL	DS	AD		Address of a 8 byte field that contains the application handle.
*				
_ACC_TRAN_ID	DS	AD		Address of a 16 byte field that contains the transaction ID.
*				
_ACC_FLAGS	DS	AD		Address of a 4 byte field that contains flags.
*				
_ACC_BUFFER4	DS	AD		Address of a data area that contains additional input data.
*				
_ACC_CLASS_CORR	DS	AD		Address of a fullword pointer that contains the address of the buffer to return the classify correlator.
*				
_ACC_END	DS	0C		End of _ACC
** BPXYWLM End				



## Appendix D. Callable services examples—AMODE 31

For an example that uses nonreentrant code, see [“Example of nonreentrant entry linkage—AMODE 31” on page 1407](#). These examples follow the rules of reentrancy. They use DSECT=NO and place the variables in the program's dynamic storage DSECT, which is allocated upon entry.

The examples are arranged alphabetically and have references to the mapping macros they use. The declaration for all local variables used in the examples follows the examples.

### Reentrant entry linkage

This entry linkage is reentrant and saves the caller's registers, allocates a save area and dynamic storage, and establishes program and dynamic storage base registers. This entry linkage is paired with the return linkage that is located at the end of the executable program; see [“Reentrant return linkage” on page 1311](#). For an example of nonreentrant entry and return linkage, see [“Example of nonreentrant entry linkage—AMODE 31” on page 1407](#).

```

        TITLE 'Alphabetical syscall of z/OS UNIX callable services'
BPXB1SM1 CSECT , Reentrant entry linkage
BPXB1SM1 AMODE 31
BPXB1SM1 RMODE ANY
        USING *,R15 Program addressability
@ENTRY0 B @ENTRY1 Branch around program header
        DROP R15 R15 not needed for addressability
        DC C'BPXB1SM1 - Reentrant callable service examples'
        DS 0H Ensure half word boundary
@ENTRY1 STM R14,R12,12(R13) Save caller's registers
        LR R2,R13 Hold address of caller's area
        LR R3,R1 Hold parameter register
        LR R12,R15 R12 program base register
        LA R11,2048(,R12) Second program base register
        LA R11,2048(,R11) Second program base register
        LA R9,2048(,R11) Third program base register
        LA R9,2048(,R9) Third program base register
        LA R4,2048(,R9) Fourth program base register
        LA R4,2048(,R4) Fourth program base register
        LA R7,2048(,R4) Fifth program base register
        LA R7,2048(,R7) Fifth program base register
        USING @ENTRY0,R12,R11,R9,R4,R7 Program addressability
        L R0,@SIZEDAT Size this program's getmain area
        GETMAIN RU,LV=(0) Getmain storage
        LR R13,R1 R13 -> this program's save area
        LA R10,2048(,R13) Second getmain base register
        LA R10,2048(,R10) Second getmain base register
        LA R6,2048(,R10) Third getmain base register
        LA R6,2048(,R6) Third getmain base register
        USING @STORE,R13,R10,R6 Getmain addressability
        ST R2,@BACK Save caller's save area pointer
        ST R13,8(,R2) Give caller our save area
        LR R1,R3 Restore parameter register
@ENTRY2 EQU * * * * * End of the entry linkage code
        SPACE ,
PSEUDO EQU * Dummy label used throughout

```

### BPX1ACC (access) example

The following code determines if `/usr/inv/network.t` can be accessed. For the callable service, see [“access \(BPX1ACC, BPX4ACC\) — Determine if a file can be accessed” on page 24](#). For the data structure, see [“BPXYACC — Map flag values for access” on page 1055](#). AMODE 64 callers use [“BPX4ACC \(access\) example” on page 1313](#).

```

MVC  BUFFERA(18),=CL18'/usr/inv/network.t'
MVC  BUFLINA,=F'18'
XC   ACC(ACC#LENGTH),ACC

```

## BPX1AC4 (access4) example

```
MVI    ACCINTENTFLAGS,ACC_R_OK+ACC_W_OK    Read and write access
SPACE ,
CALL   BPX1ACC,                             Determine accessibility of a file +
      (BUFLINA,                             Input: Pathname length           +
      BUFFERA,                             Input: Pathname                 +
      ACC,                                 Input: Access, BPXYACC          +
      RETVAL,                             Return value: 0 or -1           +
      RETCODE,                             Return code                     +
      RSNCODE),                             Reason code                     +
      VL,MF=(E,PLIST)                      -----
SPACE ,
ICM    R15,B'1111',RETVAl                Set condition code for RETVAL
BZ     PSEUDO                             Branch if RETVAL is zero
CLC    RETCODE,=A(EACCES)                 Compare RETCODE to EACCES
BE     PSEUDO                             Branch if access denied
```

## BPX1AC4 (accept4) example

The following code creates an **accept4** instance. For the callable service, use “accept4 (BPX1AC4, BPX4AC4) — Accept a connection request from a client socket and set flags” on page 16. AMODE 64 callers use the “BPX4AC4 (accept4) example” on page 1314.

```
CALL   BPX1AC4,                             Accept a socket connect request +
      (SOCKDESC,                             Input: Socket descriptor+
      =A(SOCK#LEN+SOCK_SUN#LEN),             Output: Write file descriptor +
      SOCADDR,                             Input: Sockaddr structure +
      FLAGS,                               Input: flag
      RETVAL,                             Return value: 0 or -1 +
      RETCODE,                             Return code +
      RSNCODE),                             Reason code +
      VL,MF=(E,PLIST)
```

## BPX1ACK (auth\_check\_resource\_np) example

The following code determines if user JOEUSER has UPDATE access to the FACILITY class profile TEST.THIS.PROFILE. For the callable service, see “auth\_check\_resource\_np (BPX1ACK, BPX4ACK) — Determine a user's access to a RACF-protected resource” on page 64. AMODE 64 callers use “BPX4ACK (auth\_check\_resource\_np) example” on page 1314.

```
MVI    CELLUUID,X'00'
MVI    PRINUUID,X'00'
MVC     USERNLEN,=F'7'
MVC     USERNAME(7),=CL7'JOEUSER'
MVC     CLSLEN,=F'8'
MVC     CLS(8),=CL8'FACILITY'
MVC     ENTLEN,=F'17'
MVC     ENT(17),=CL17'TEST.THIS.PROFILE'
SPACE ,
CALL   BPX1ACK,                             Determine access to a resource +
      (CELLUUID,                             Input: Cell UUID             +
      PRINUUID,                             Input: Principal UUID        +
      USERNLEN,                             Input: Userid length         +
      USERID,                              Input: Userid                +
      CLSLEN,,                             Input: Class length          +
      CLS,                                 Input: Class                  +
      ENTLEN,                             Input: Entity length         +
      ENT,                                Input: Entity                +
      =A(ACK_UPDATE#),                     Input: Access type to check for +
      RETVAL,                             Return value: 0 or -1         +
      RETCODE,                             Return code                   +
      RSNCODE),                             Reason code                   +
      VL,MF=(E,PLIST)                      -----
```

01

## BPX1ACP (accept) example

The following code does an accept to accept a connect request from a client. SOCKDESC was previously set by a call to BPX1SOC. A bind and a listen must also have been previously done. The SOCKADDR

was built by the call to BPX1BND. For the callable service, see [“accept \(BPX1ACP, BPX4ACP\) — Accept a connection request from a client socket”](#) on page 13. For the data structure, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151. AMODE 64 callers use [“BPX4ACP \(accept\) example”](#) on page 1314.

```
CALL BPX1ACP,          Accept a socket connect request +
    (SOCKDESC,        Input: Socket descriptor +
    =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr +
    SOCKADDR,         Input: Sockaddr structure +
    RETVAL,           Return value: 0 or -1 +
    RETCODE,          Return code +
    RSNCODE),         Reason code +
    VL,MF=(E,PLIST)   -----
L   R2,RETVAL
ST  R2,SOCKDES2       Store the new socket descriptor
```

## BPX1AC4 (accept4) example

The following code creates an **accept4** instance. For the callable service, use [“accept4 \(BPX1AC4, BPX4AC4\) — Accept a connection request from a client socket and set flags”](#) on page 16. AMODE 64 callers use the [“BPX4AC4 \(accept4\) example”](#) on page 1314.

```
CALL BPX1AC4,          Accept a socket connect request +
    (SOCKDESC,        Input: Socket descriptor+
    =A(SOCK#LEN+SOCK_SUN#LEN), Output: Write file descriptor +
    SOCADDR,         Input: Sockaddr structure +
    FLAGS,           Input: flag +
    RETVAL,          Return value: 0 or -1 +
    RETCODE,         Return code +
    RSNCODE),        Reason code +
    VL,MF=(E,PLIST)
```

## BPX1AIO (asyncio) example

The following code will accept the next conversation. For the callable service, see [“asyncio \(BPX1AIO, BPX4AIO\) — Asynchronous I/O for sockets”](#) on page 32. AMODE 64 callers use [“BPX4AIO \(asyncio\) example”](#) on page 1315.

```
XC   AIO(AIO#LENGTH),AIO  Null AIO control block
MVC  AIOCMD,=A(AIO#ACCEPT) Command = Accept
MVC  AIOFD,FILEDESC      File descriptor
MVC  AIONOTIFYTYPE,=AL2(AIO#MVS) Notify type = MVS
XC   ECB01,ECB01         ECB = 0
LA   R15,ECB01           ECB Address
ST   R15,AIOECBPTR       Null AIO control block
MVC  AIOSOCKADDRLEN,=A(SOCK#LEN)
LA   R15,SOCKADDR        From recvform (see BPX1RFM)
ST   R15,AIOSOCKADDRPTR
SPACE
CALL BPX1AIO,            Asynchronous I/O for Sockets +
    (=A(AIO#LENGTH),    Input: Time before SIGAIOM +
    AIO,                Input: Time before SIGAIOM +
    RETVAL,             Return value: 0 or -1 +
    RETCODE,            Return code +
    RSNCODE),           Reason code +
    VL,MF=(E,PLIST)     -----
```

## BPX1ALR (alarm) example

The following code schedules an alarm in 5 seconds. For the callable service, see [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm”](#) on page 30. AMODE 64 callers use [“BPX4ALR \(alarm\) example”](#) on page 1315.

```
MVC  SECONDS,=F'5'
SPACE ,
```

## BPX1ANR (accept\_and\_recv) example

```
CALL BPX1ANR,      Schedule Alarm      +
      (SECONDS,    Input: Time before SIGALRM  +
      RETVAL),     Return value: 0 or -1      +
      VL,MF=(E,PLIST) -----
```

## BPX1ANR (accept\_and\_recv) example

The following code accepts a connection and reads the first block of data from a client. The new socket's descriptor, the peer's remote address and the caller's local address are also returned. SOCKDESC was previously set by a call to BPX1SOC. ACPSOCK must be set to -1 and the system will assign a new descriptor for the accepted connection in this parameter. A bind and a listen must also have been previously done. The SOCKADDR was built by the call to BPX1BND. For the callable service, see [“accept\\_and\\_recv \(BPX1ANR, BPX4ANR\) — Accept a connection and receive the first block of data” on page 19](#). For the data structure, see [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#). AMODE 64 callers use [“BPX4ANR \(accept\\_and\\_recv\) example” on page 1316](#).

```
      L      R8,=XL4'FFFFFFFF'      Set ACPSOCK = -1
      ST      R8,ACPSOCK
      CALL    BPX1ANR,              Accept_and_receive request  +
      (SOCKDESC,                    Input: Socket descriptor    +
      ACPSOCK,                      Input: -1 Output: accepted soc des+
      SOCK#LEN+SOCK_SUN#LEN,        Input/Output: Len of Remote_addr +
      RSOCKADR,                    Input: Remote sockaddr structure +
      SOCK#LEN+SOCK_SUN#LEN,        Input/Output: Len of Local_addr  +
      LSOCKADR,                    Input: Local sockaddr structure  +
      =A(L'BUFFERA),               Input: Length of the buffer     +
      BUFFERA,                    Input/Output: Addr of the buffer  +
      PRIMARYALET,                 Input: Alet of the buffer       +
      RETVAL,                      Return value: -1 or num bytes recd+
      RETCODE,                    Return code                     +
      RSNCODE),                   Reason code                     +
      VL,MF=(E,PLIST) -----
      L      R2,RETVAL
      ST      R2,BYTERECD           Store number of bytes received
```

## BPX1ASP (aio\_suspend) example

The following code will wait up to 10 seconds for one of the events specified in the AIOCB. For the callable service, see [“aio\\_suspend \(BPX1ASP, BPX4ASP\) — Wait for an asynchronous I/O request” on page 27](#). AMODE 64 callers use [“BPX4ASP \(aio\\_suspend\) example” on page 1316](#).

```
      LA      R15,AIO
      ST      R15,ARGSLST
      MVC     ARGCNT,=F'1'
      MVC     SECONDS,=F'10'
      XC      NANOSECONDS,NANOSECONDS
      SPACE   ,
      CALL    BPX1ASP,              Suspend for an aio request  +
      (ARGSLST,                    Input: List of pointers to AIOCBs +
      ARGCNT,                      Input: Count of pointers in list  +
      SECONDS,                    Input: Seconds to wait             +
      NANOSECONDS,                Input: Nanoseconds to wait         +
      RETVAL,                    Return value: 0 or -1              +
      RETCODE,                    Return code                       +
      RSNCODE),                   Reason code                       +
      VL,MF=(E,PLIST) -----
```

## BPX1ATM (attach\_execmvs) example

The following code invokes program APPL92 on a subtask and as a child process of the caller, passing the length and parameter MONTH9,PRELIM,(232/74.99). There is no exit routine that is associated with program APPL92. For the callable service, see [“attach\\_execmvs \(BPX1ATM, BPX4ATM\) — Attach an MVS program” on page 58](#). AMODE 64 callers use [“BPX4ATM \(attach\\_execmvs\) example” on page 1316](#).

```
      MVC     PGMNAMEL,=F'6'
```

```

MVC  PGMNAME(06),=CL6'APPL92'
MVC  BUFLNA,=F'24'
MVC  BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
SPACE ,
CALL  BPX1ATM,          Invoke a MVS program          +
      (PGMNAMEL,        Input: Length of program name  +
      PGMNAME,          Input: Program name            +
      BUFLNA,           Input: Length of program argument +
      BUFFERA,          Input: Program argument        +
      =A(0),            Input: Exit routine address or 0 +
      =A(0),            Input: Exit Parm list address or 0+
      RETVAL,           Return value: Child PID Or -1    +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      VL,MF=(E,PLIST)   -----

```

## BPX1ATX (attach\_exec) example

The program ictasma located at ict/bin gets control on a subtask and as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see [“attach\\_exec \(BPX1ATX, BPX4ATX\) — Attach a z/OS UNIX program”](#) on page 48. AMODE 64 callers use [“BPX4ATX \(attach\\_exec\) example”](#) on page 1317.

```

MVC  BUFLNA,=F'16'
MVC  BUFFERA(16),=C'/ict/bin/ictasma'
MVC  ARGCNT,=F'3'
*
*      First
LA    R15,=F'4'          Length
ST    R15,ARGLLST+00     Length parm list
LA    R15,=CL4'WK18'     Argument
ST    R15,ARGSLST+00     Argument address parm list
*
*      Second
LA    R15,=F'7'          Length
ST    R15,ARGLLST+04     Length parm list
LA    R15,=CL7'DEPT37A'  Argument
ST    R15,ARGSLST+04     Argument address parm list
*
*      Third
LA    R15,=F'22'         Length
ST    R15,ARGLLST+08     Length parm list
LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)' Argument
ST    R15,ARGSLST+08     Argument address parm list
*
MVC  ENVCNT,=F'0'        Number of env. data items passed
MVC  ENVLENS,=F'0'       Addr of env. data length list
MVC  ENVPARMS,=F'0'      Add of env. data
*
MVC  EXITRTNA,=V(EXITRTN) ->exit routine
*
MVC  EXITPLA,=A(exit paramter list as expected by EXITRTN)
SPACE ,
CALL  BPX1ATX,          +
      (BUFLNA,          Input: Pathname length        +
      BUFFERA,          Input: Pathname                +
      ARGCNT,           Input: Argument count          +
      ARGLLST,          Input: Argument length list    +
      ARGSLST,          Input: Argument address list    +
      ENVCNT,           Input: Environment count       +
      ENVLENS,          Input: Environment length list  +
      ENVPARMS,         Input: Environment address list +
      EXITRTNA,         Input: Exit routine address or 0 +
      EXITPLA,          Input: Exit Parm list address or 0+
      RETVAL,           Return value: Child PID or -1    +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      VL,MF=(E,PLIST)   -----

```

## BPX1AUT (\_\_authenticate) example

The following code authenticates a user with username and password and generates an identity token. For the callable service, use [“\\_\\_authenticate \(BPX1AUT, BPX4AUT\) — Authenticate the specified user's credentials”](#) on page 69. AMODE 64 callers use the [“BPX4AUT \(\\_\\_authenticate\) example”](#) on page 1317.

```

MVC  USERLEN,=F'8'

```

## BPX1BND (bind) example

```
MVC  USERNAME(8),=CL8'Myuserid'
MVC  OLDPASSLEN,=F'8'
MVC  OLDPASS(8),=CL8'MyOldPwd'
MVC  BUFLINA,=F'1024'
MVC  OPTIONS,=A(AUTH_BUILD_IDT#)
SPACE ,
CALL  BPX1AUT,          authenticate users credentials      +
      (=A(AUTH_USER_ID#), Input: Type of credentials      +
      USERNLEN,          Input/Output: User ID length      +
      USERNAME,          Input/Output:                    +
      OLDPASSLEN,        Input: Pass_length                +
      OLDPASS,           Input: Pass                      +
      =A(0),             Input: New_pass_length            +
      NEWPASS,           Input: New_pass                   +
      BUFLINA,           Input: Idt_buffer_length          +
      BUFFERA,           Input: Idt_buffer_ptr             +
      IDTLEN,            Input/Output: Idt_length          +
      MSGBUFSP,          Input: Msg_buffer_subpool         +
      MSGBUFPTR,         Output: Msg_buffer_ptr            +
      =A(0),             Input: Appl_id_length             +
      APPLID,            Input: Appl_id                    +
      OPTIONS,           Input/OUTPUT: Option_Flags        +
      RETVAL,            Return value 0 or -1              +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                      +
      VL,MF=(E,PLIST)   -----
```

## BPX1BND (bind) example

The following code does a bind to associate a name with a socket. SOCKDESC was previously set by a call to BPX1SOC. For the callable service, see “bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor” on page 76. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 64 callers use “BPX4BND (bind) example” on page 1318.

```
SPACE ,
MVI  SOCK_LEN,12        Store the length of the address
MVI  SOCK_FAMILY,AF_UNIX Set the domain to AF_UNIX
MVC  SOCK_SUN_NAME(12),=CL12'/tmp/socket1' Set the name
CALL  BPX1BND,          Bind a name to a socket            +
      (SOCKDESC,        Input: Socket Descriptor          +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr +
      SOCKADDR,         Input: Sockaddr structure         +
      RETVAL,           Return value: 0 or -1              +
      RETCODE,          Return code                      +
      RSNCODE),         Reason code                      +
      VL,MF=(E,PLIST)   -----
```

## BPX1BAS (bind with source address selection) example

The following code does a bind to associate the best source address for the provided destination IP address with a socket. SOCKDESC was previously set by a call to BPX1SOC. For the callable service, see “bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address” on page 79. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151.

```
SPACE ,
MVI  SOCK_LEN,=A(SOCK#LEN+SOCK_SIN6#LEN) Store the length of the address
MVI  SOCK_FAMILY,AF_INET6 Set the domain to AF_INET6
MVC  SOCK_SIN6_ADDR,=XL16'00A100B200C300D400E500F61234ABCD'
CALL  BPX1BAS,          Bind with source address selection+
      (SOCKDESC,        Input: Socket Descriptor          +
      =A(SOCK#LEN+SOCK_SIN6#LEN), Input:Length - Sockaddr +
      SOCKADDR,         Input: Sockaddr structure         +
      RETVAL,           Return value: 0 or -1              +
      RETCODE,          Return code                      +
      RSNCODE),         Reason code                      +
      VL,MF=(E,PLIST)   -----
```

## BPX1CCA (cond\_cancel) example

The following code cancels a program's interest in events that were selected by a call to the cond\_setup service. For the callable service, see “cond\_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events” on page 125. AMODE 64 callers use “BPX4CCA (cond\_cancel) example” on page 1318.

```

CALL BPX1CCA,          Cancel cond_setup          +
   (RETVAL,           Return value: 0 or -1        +
    RETCODE,          Return code                  +
    RSNCODE),         Reason code                  +
    VL,MF=(E,PLIST)   -----
* The return value (RETVAL) does not matter. When your program
* receives control following the call to cond_cancel, it is no
* longer eligible to receive event notifications via cond_post.

```

## BPX1CCS (\_\_console()) example

The following code sends a message to the console. For the callable service, see “\_\_console() (BPX1CCS, BPX4CCS) — Communicate with the console ” on page 142. For the data structure, see “BPXYCCA — Map input/output structure for \_\_console()” on page 1060. AMODE 64 callers use “BPX4CCS (\_\_console()) example” on page 1319.

```

CALL BPX1CCS,          Send msg to console          +
   (MSGATTRLEN,       Input: BPXYCCA length         +
    MSGATTR,          Input: BPXYCCA                +
    MODSTRINGPTR,     Output: Modify msg from console +
    MODIFYSTGLLEN,    Output: Length of modify msg   +
    CONMSGTYPE,       Output: Console msg type       +
    RETVAL,           Return value: 0 or -1          +
    RETCODE,          Return code                    +
    RSNCODE),         Reason code                    +
    VL,MF=(E,PLIST)   -----

```

## BPX1CHA (chaudit) example

The following code changes the audit flags for the file identified by *pathname*. For the callable service, see “chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path ” on page 95. For the data structure, see “BPXYAUDT — Map flag values for chaudit and fchaudit” on page 1060. AMODE 64 callers use “BPX4CHA (chaudit) example” on page 1319.

```

MVC  BUFFERA(18),=CL18'/usr/inv/network.t'
MVC  BUFLINA,=F'18'
MVI  AUDTREADACCESS,AUDTREADFAIL
MVI  AUDTWRITEACCESS,AUDTWRITEFAIL
MVI  AUDTEXECACCESS,AUDTEXECFAIL
MVI  AUDTRSRV,0
SPACE
CALL BPX1CHA,          Change audit                +
   (BUFLINA,          Input: Pathname length        +
    BUFFERA,          Input: Pathname                +
    AUDT,             Input: Audit flags, BPXYAUDT   +
    =F'0',            Input: 0 user, 1 security auditor +
    RETVAL,           Return value: 0 or -1          +
    RETCODE,          Return code                    +
    RSNCODE),         Reason code                    +
    VL,MF=(E,PLIST)   -----

```



## BPX1CHD (chdir) example

The following code changes the working directory for the task. For the callable service, see “chdir (BPX1CHD, BPX4CHD) — Change the working directory” on page 98. AMODE 64 callers use “BPX4CHD (chdir) example” on page 1319.

```

MVC  BUFFERA(8),=CL8'/usr/inv'
MVC  BUFLINA,=F'8'
SPACE ,
CALL  BPX1CHD,          Change working directory          +
      (BUFLINA,          Input: Pathname length           +
      BUFFERA,           Input: Pathname                  +
      RETVAL,            Return value: 0 or -1             +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                      +
      VL,MF=(E,PLIST)   -----

```

## BPX1CHM (chmod) example

The following code changes the file mode for the file identified by pathname. For the callable service, see “chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory” on page 101. For the data structure, see “BPXYMODE — Map the mode constants” on page 1106. AMODE 64 callers use “BPX4CHM (chmod) example” on page 1319.

```

MVC  BUFFERA(26),=CL26'newprogs/path/eightfold.c'
MVC  BUFLINA,=F'26'
XC   S_MODE,S_MODE
MVI  S_MODE2,S_IRUSR      All read and write
MVI  S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL  BPX1CHM,          Change File Modes                  +
      (BUFLINA,          Input: Pathname length           +
      BUFFERA,           Input: Pathname                  +
      S_MODE,            Input: Mode, mapped by BPXYMODE  +
      RETVAL,            Return value: 0 or -1             +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                      +
      VL,MF=(E,PLIST)   -----

```

## BPX1CHO (chown) example

The following code changes the owner of /somedir/somefile.c from the current owner to that specified by USERID and GROUPID. For the callable service, see “chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory” on page 104. AMODE 64 callers use “BPX4CHO (chown) example” on page 1320.

```

MVC  BUFFERA(20),=CL20'/somedir/somefile.c'
MVC  BUFLINA,=F'20'
MVC  USERID,..          New owner UID from stat           07
MVC  GROUPID,..         New owner GID from stat           07
SPACE ,
CALL  BPX1CHO,          Change owner and group of a file  +
      (BUFLINA,          Input: Pathname length           +
      BUFFERA,           Input: Pathname                  +
      USERID,            Input: New owner UID              +
      GROUPID,           Input: New owner GID              +
      RETVAL,            Return value: 0 or -1             +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                      +
      VL,MF=(E,PLIST)   -----

```

## BPX1CHOA (fchownat) example

The following code changes ownership of a file relative to a directory file descriptor. For the callable service, use “fchownat (BPX1COA, BPX4COA) — Update the file ownership of a file or directory relative to



a directory file descriptor” on page 213. AMODE 64 callers use the “BPX4CHOA (fchownat) example” on page 1320.

```

MVC FILEDESC,..           From opendir
MVC GROUPID,..           Group ID 15
MVC USERID,..           User ID 15
SPACE ,
CALL BPX1CHOA,           Change the owner and group of file+
(FILEDESC,              Input: Directory File descriptor +
USERID,                 Input: New user ID for file      +
GROUPID,               Input: New group ID for file      +
FLAGS,                 Input: Flags                      +
RETV,                  Return value: 0 or -1             +
RETCODE,               Return code                      +
RSNCODE),              Reason code                      +
VL,MF=(E,PLIST)         -----

```

## BPX1CHP (chpriority) example

The following code changes the CPU priority based on the input which, who, and priority type values. The which value used is PRIO\_PROCESS, indicating that the priority will be set by process ID. The who value used is 7, to set the priority for process ID 7. The priority type is CPRIQ\_ABSOLUTE, indicating that the priority will be set to the value specified, 1. For the callable service, see “chpriority (BPX1CHP, BPX4CHP) — Change the scheduling priority of a process” on page 107. AMODE 64 callers use “BPX4CHP (chpriority) example” on page 1320.

```

MVC PROCID,=XL4'00000007' Process ID to change priority for
MVC PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL BPX1CHP,           Change priority value          +
(=A(PRIO_PROCESS),      Input: Set by Process ID       +
PROCID,                Input: PID to set priority for  +
=A(CPRIQ_ABSOLUTE),     Input: Change by absolute value +
PRIORITY,              Input: Priority value to change to+
RETV,                  Return value: 0 or -1           +
RETCODE,               Return code                    +
RSNCODE),              Reason code                    +
VL,MF=(E,PLIST)         -----
L   R15,RETV            Load return value
C   R15,=F'-1'          Test for -1 return
BE  PSEUDO              Branch on error

```

## BPX1CMA (fchmodat) example

The following code changes the file mode for the file identified by Pathname within the directory represented by the file descriptor FILEFD. Prior to this code excerpt, FILEFD would have been to the file descriptor of the specified directory or to AT\_FDCWD to indicate the current directory. For the callable service, see “fchmodat (BPX1CMA, BPX4CMA) — Change the mode of a file or directory relative to a directory file descriptor” on page 206. For the data structure, see “BPXYMODE — Map the mode constants” on page 1106. AMODE 64 callers use the “BPX4CMA (fchmodat) example” on page 1321.

```

*
*   Issue fchmodat
*
MVC BUFLINA,=F'6'
MVC BUFFERA(6),=CL6'test.c'
XC  S_MODE,S_MODE
MVI S_MODE2,S_IRUSR
XC  ATFLAGS,ATFLAGS
MVI ATFLAGS,AT_SYMLINK_NOFOLLOW Do not follow symlinks
MVI S_MODE3,S_IRWXU2+S_IRGRP+S_IROTH
*
*   Permissions -
*   User All (RWX), Grp & Other Read
CALL BPX1CMA,           Change File Mode - fchmodat    +
(BUFLINA,              Input: Pathname length          +

```

## BPX1CHR (chattr) example

	BUFFERA,	Input: Pathname	+
	S_MODE,	Input: Mode, mapped by BPXYMODE	+
	FILEDESC,	Input: Directory file desc	+
	ATFLAGS	Input: At flags	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)		
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	
LR	R4,R15	Save the return value	

## BPX1CHR (chattr) example

The following code changes the attributes of /somedir/somefile.c. The owning user and group IDs are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory”](#) on page 82. For the data structures, see [“BPXYATT — Map file attributes for chattr and fchattr”](#) on page 1057 and [“BPXYMODE — Map the mode constants”](#) on page 1106. AMODE 64 callers use [“BPX4CHR \(chattr\) example”](#) on page 1321.

```
MVC BUFFERA(20),=CL20'/somedir/somefile.c'
MVC BUFLINA,=F'20'
MVC ATTID,=CL4'ATT '      Eye Catcher
MVC ATTVERSION,=AL2(ATT#VER) version
XC S_MODE,S_MODE          Clear mode
MVI S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC ATTMODE,S_MODE        Move mode data to attribute structure
MVC ATTUID,=F'7'          Specify new UID
MVC ATTGID,=F'77'         Specify new GID
OI ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG      +
Flag Mode, UID and GID changes
OI ATTSETFLAGS2,ATTCTIMETOD                  +
Set change time to current time
SPACE ,
CALL BPX1CHR,                                +
      (BUFLINA,                               +
      BUFFERA,                               +
      =A(ATT#LENGTH),                        +
      ATT,                                   +
      RETVAL,                               +
      RETCODE,                              +
      RSNCODE),                             +
      VL,MF=(E,PLIST)                       -----
```

## BPX1CLN (clone) example

The following code clones a process that will be associated with a newly created mount namespace. The CLNP structure is filled in with the request information. The next sequential instruction obtains control from both the parent process (RETVAL=*child process ID*) and from the child process (RETVAL=0). If RETVAL=-1, the clone failed.

For the callable service, see [clone \(BPX1CLN, BPX4CLN\) — Create a new process](#). For the data structure, see [“BPXYCLNP— Map clone syscall parameters”](#) on page 1062 and [“BPXYSIGH — Signal constants”](#) on page 1148. For the AMODE 64 example, see [“BPX4CLN \(clone\) example”](#) on page 1322.

```
LA R5,CLN_PARM
USING CLNP,R5
XC CLNP(CLNP#LEN),CLNP      Clear the input structure
MVC CLNP_ID,=A(CLNP#ID)    CLNP identifier
MVHHI CLNP_VER,CLNP#VER    Current CLNP version
MVHHI CLNP_LEN,CLNP#LEN    CLNP length
MVC CLNP_FLAGS,=A(CLONE_NEWNS) New mount namespace
```

```

MVC CLNP_EXIT_SIGNAL,=A(SIGCHLD#) Must be SIGCHLD
CALL BPX1CLN,
      (=A(CLN#LEN),      Length of clone parameters      +
      CLN_PARM,          Clone parameters                +
      RETVAL,            Return value: -1, 0, child's PID  +
      RETCODE,           Return code                     +
      RSNCODE),          Reason code                     +
      MF=(E,PLIST)
L    R15,RETVAL          Load return value
C    R15,=F'-1'          Test for -1 return
BE   PSEUDO              Branch on error

```

## BPX1CLD (closedir) example

The following code closes the directory identified by FILEDESC. For the callable service, see “[closedir \(BPX1CLD, BPX4CLD\) — Close a directory](#)” on page 124. AMODE 64 callers use “[BPX4CLD \(closedir\) example](#)” on page 1322.

```

MVC FILEDESC,..        Directory descriptor from opendir 08
SPACE ,
CALL BPX1CLD,          Close a directory                +
      (FILEDESC,       Input: Directory file descriptor  +
      RETVAL,          Return value: 0 or -1             +
      RETCODE,         Return code                     +
      RSNCODE),        Reason code                     +
      VL,MF=(E,PLIST)  -----

```

## BPX1CLO (close) example

The following code closes the standard input file. For the callable service, see “[close \(BPX1CLO, BPX4CLO\) — Close a file](#)” on page 121. AMODE 64 callers use “[BPX4CLO \(close\) example](#)” on page 1323.

```

CALL BPX1CLO,          Close a file                    +
      (=A(STDIN_FILENO), Input: File descriptor        +
      RETVAL,          Return value: 0 or -1           +
      RETCODE,         Return code                     +
      RSNCODE),        Reason code                     +
      VL,MF=(E,PLIST)  -----

```

## BPX1CON (connect) example

The following code connects to a socket. SOCKDESC was returned by a previous call to BPX1SOC, and SOCKADDR contains the name of the peer, possibly obtained by a call to BPX1GNM. For the callable service, see “[connect \(BPX1CON, BPX4CON\) — Establish a connection between two sockets](#)” on page 139. For the data structure, see “[BPXYSOCK — Map SOCKADDR structure and constants](#)” on page 1151. AMODE 64 callers use “[BPX4CON \(connect\) example](#)” on page 1323.

```

SPACE ,
MVI  SOCK_LEN,12        Store the length of the address
MVI  SOCK_FAMILY,AF_UNIX Set the domain to AF_UNIX
MVC  SOCK_SUN_NAME(12),=CL12'/tmp/socket1' Set the name
CALL BPX1CON,           Connect to a socket            +
      (SOCKDESC,        Input: Socket Descriptor      +
      SOCK#LEN+SOCK_SUN#LEN, Input: Length - Sockaddr  +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr +
      RETVAL,          Return value: 0 or -1           +
      RETCODE,         Return code                     +
      RSNCODE),        Reason code                     +
      VL,MF=(E,PLIST)  -----

```

## BPX1CPO (cond\_post) example

The following code demonstrates how to send an event notification to a thread waiting in the cond\_wait or cond\_timed\_wait service. For the callable service, see “cond\_post (BPX1CPO, BPX4CPO) — Post a thread for an event” on page 127. AMODE 64 callers use “BPX4CPO (cond\_post) example” on page 1323. The following code notifies thread (THID) that a CW\_CONDVAR event has occurred.

CALL	BPX1CPO,	Send condition event notification +
	(THID,	Input: Thread ID of target pgm +
	=A(CW_CONDVAR),	Input: Event in BPXYCW +
	RETVAL,	Return value: 0 or -1 +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	VL,MF=(E,PLIST)	-----

## BPX1CRA (chatrat) example

The following code modifies the attributes that are associated with a file. For the callable service, see “chatrat (BPX1CRA, BPX4CRA) — Change the attributes of a file or directory relative to a directory file descriptor” on page 91. For the data structures, see “BPXYATT — Map file attributes for chatr and fchatr” on page 1057 and “BPXYMODE — Map the mode constants” on page 1106. AMODE 64 callers use “BPX4FCR (fchatr) example” on page 1331.

MVC	FILEDESC,...	File descriptor
MVC	BUFFERA(20),=CL20'/somedir/somefile.c'	
MVC	BUFLINA,=F'20'	
MVC	ATTID,=CL4'ATT '	Eye Catcher
MVC	ATTVERSION,=AL2(ATT#VER)	version
XC	S_MODE,S_MODE	Clear mode
MVI	S_MODE2,S_IRUSR	Read-execute/write/read-execute
MVI	S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH	
MVC	ATTMODE,S_MODE	Move mode data to attribute + structure
MVC	ATTUID,=F'7'	Specify new UID
MVC	ATTGID,=F'77'	Specify new GID
OI	ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG	
+		
		Flag Mode, UID and GID changes
OI	ATTSETFLAGS2,ATTCTIMETOD	
+		
		Set change time to current time
SPACE		
CALL	BPX1CRA,	Change file attributes +
	(FILEDESC,	Input: File descriptor +
	BUFLINA,	Input: Pathname length +
	BUFFERA,	Input: Pathname +
	=A(ATT#LENGTH),	Input: BPXYATT length +
	ATT,	Input/output: BPXYATT +
	FLAGS,	Input: Flags +
	RETVAL,	Return value: 0 or -1 +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	VL,MF=(E,PLIST)	-----

## BPX1CRT (chroot) example

The following code changes the root directory for the task. For the callable service, see “chroot (BPX1CRT, BPX4CRT) — Change the root directory” on page 111. AMODE 64 callers use “BPX4CRT (chroot) example” on page 1324.

MVC	BUFFERA(8),=CL8'/usr/inv'	
MVC	BUFLINA,=F'8'	
SPACE		
CALL	BPX1CRT,	Change root directory +
	(BUFLINA,	Input: Pathname length +
	BUFFERA,	Input: Pathname +
	RETVAL,	Return value: 0 or -1 +
	RETCODE,	Return code +

RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1CRT (chroot) example

The following code changes the root directory for the task. For the callable service, see “chroot (BPX1CRT, BPX4CRT) — Change the root directory” on page 111. AMODE 64 callers use “BPX4CRT (chroot) example” on page 1324.

MVC	BUFFERA(8),=CL8'/usr/inv'		
MVC	BUFLINA,=F'8'		
SPACE	,		
CALL	BPX1CRT,	Change root directory	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1CSE (cond\_setup) example

The following code sets up the invoker to suspend processing until any of the specified events (CW\_INTRPT or CW\_CONDDVAR) occurs. The BPX1CTW (cond\_timed\_wait) or BPX1CWA (cond\_wait) service is used to actually suspend processing. For the callable service, see “cond\_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications ” on page 130. AMODE 64 callers use “BPX4CSE (cond\_setup) example” on page 1324.

MVC	EVENTLIST,=A(CW_INTRPT+CW_CONDDVAR)		
CALL	BPX1CSE,	Condition setup	+
	(EVENTLIST,	Input: Event list	BPXYCW +
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1CTW (cond\_timed\_wait) example

The following code suspends the calling thread until a signal arrives (CW\_INTRPT), or else 2.5 seconds have elapsed. For the callable service, see “cond\_timed\_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event ” on page 132. AMODE 64 callers use “BPX4CTW (cond\_timed\_wait) example” on page 1325.

MVC	EVENTLIST,=A(CW_INTRPT)	Signals	
CALL	BPX1CTW,	Wait for condition events	+
	(=A(2),	Input: Number of seconds	+
	=A(5000000000),	Input: Number of nanoseconds	+
	EVENTLIST,	Input: Event list	BPXYCW +
	SECONDS,	Output: Unexpired seconds	+
	NANOSECONDS,	Output: Unexpired nanoseconds	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1CWA (cond\_wait) example

The following code suspends the calling thread until either of two events occurs: the arrival of a signal (CW\_INTRPT) or some other thread using the cond\_post service to send this thread a CW\_CONDDVAR

## BPX1DEL (deleteHFS) example

notification. For the callable service, see [“cond\\_wait \(BPX1CWA, BPX4CWA\) — Suspend a thread for an event”](#) on page 136. AMODE 64 callers use [“BPX4CWA \(cond\\_wait\) example”](#) on page 1325.

```
MVC  EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL BPX1CWA,          Wait for condition events      +
      (EVENTLIST,      Input: Event list             BPXYCW +
      RETVAL,          Return value: 0 or -1          +
      RETCODE,         Return code                   +
      RSNCODE),        Reason code                   +
      VL,MF=(E,PLIST)  -----
```

## BPX1DEL (deleteHFS) example

The program ictasma located at ict/bin is loaded into storage using BPX1LOD, branched to and then deleted from storage using BPX1DEL. For the callable service, see [“deletehfs \(BPX1DEL, BPX4DEL\) — Delete a program from storage”](#) on page 148. AMODE 64 callers use [“BPX4DEL \(deleteHFS\) example”](#) on page 1325.

```
MVC  BUFLINA,=F'16'
MVC  BUFFERA(16),=C'/ict/bin/ictasma'
MVC  OPTIONS,=A(0)
MVC  LIBPHTLN,=A(0)
SPACE ,
CALL BPX1LOD,          Load Program                  +
      (BUFLINA,        Input: Pathname length        +
      BUFFERA,         Input: Pathname               +
      OPTIONS,         Input: Options                +
      LIBPHTLN,        Input: Library Path Length    +
      LIBPATH,         Input: Library Path           +
      EPADDR,          Return value: -1 or entry pt addr +
      RETCODE,         Return code                   +
      RSNCODE),        Reason code                   +
      VL,MF=(E,PLIST)  -----
L     R15,EPADDR       Load return value
C     R15,=F'-1'       Test for -1 return
BE    PSEUDO           Branch on error
SPACE ,
L     R15,EPADDR
BALR  R14,R15          Branch to loaded program
SPACE ,
CALL BPX1DEL,          Delete program                +
      (EPADDR,         Input: Entry point address    +
      RETVAL,          Return value: -1 or 0          +
      RETCODE,         Return code                   +
      RSNCODE),        Reason code                   +
      VL,MF=(E,PLIST)  -----
```

## BPX1EFD (eventfd) example

The following code creates an **eventfd** instance. For the callable service, use [“eventfd \(BPX1EFD, BPX4EFD\) — Create a file descriptor for event notification”](#) on page 160. AMODE 64 callers use the [“BPX4EFD \(eventfd\) example”](#) on page 1326.

```
MVI  INITVAL,1,        Set Initial count to 1
XC   FLAGS(4),FLAGS
MVI  FLAGS,EFD_SEMAPHORE, READ with SEMAPHORE
SPACE ,
CALL BPX1EFD,          Create a eventfd              +
      (INITVAL,        Input: Initial count          +
      FLAGS,           Input: eventfd Flags          BPXYEFD +
      RETVAL,          Return value: 0 or -1          +
      RETCODE,         Return code                   +
      RSNCODE),        Reason code                   +
      VL,MF=(E,PLIST)  -----
ICM  R15,B'1111',RETVAL Test RETVAL
BL   PSEUDO            Branch if negative (-1 = failure)
ST   R15,FILEDESC      Store the file descriptor
```

## BPX1ENV (oe\_env\_np) example

The following code enables interruption of threads waiting in MVS ENQs in the caller's process. For the callable service, see [“oe\\_env\\_np \(BPX1ENV, BPX4ENV\) — Examine, change, or examine and change an environmental attribute”](#) on page 500. For the data structure, see [“BPXYCONS — Constants used by services”](#) on page 1062. AMODE 64 callers use [“BPX4ENV \(oe\\_env\\_np\) example”](#) on page 1326.

```

LA      R15,=F'1'
ST      R15,INARG
LA      R15,INARG
ST      R15,INARGLIST
LA      R15,INARGLIST
ST      R15,INARGLISTPTR
SPACE ,
CALL    BPX1ENV,          oe_env_np          +
      (=A(ENQWAIT_PROCESS), Input: Function_code    BPXYCONS +
      =A(1),              Input: InArgCount          +
      INARGLISTPTR,       Input: InArgListPtr        +
      =A(0),              Input: OutArgCount         +
      =A(0),              Input: OutArgListPtr       +
      RETVAL,             Return value: 0 or -1      +
      RETCODE,            Return code                +
      RSNCODE),           Reason code                +
      VL,MF=(E,PLIST)    -----

```

## BPX1EPC (epoll\_create) example

The following code creates an **epoll\_create** instance. For the callable service, use [“epoll\\_create \(BPX1EPC, BPX4EPC\) — Open an epoll file descriptor”](#) on page 150. AMODE 64 callers use the [“BPX4EPC \(epoll\\_create\) example”](#) on page 1326. For the data structure, see [“BPXYEPOL— Map epoll syscall parameters”](#) on page 1075.

```

XC      OPTION,OPTION
CALL    BPX1EPC,          Create an epoll file
      (OPTION,            Input: Option
      RETVAL,             Return value: EPOLL_FD or -1
      RETCODE,            Return code
      RSNCODE),           Reason code
      VL,MF=(E,PLIST)

```

## BPX1EPL (epoll\_epl) example

The following code creates an **epoll\_epl** instance. For the callable service, use [“BPX1EPL \(epoll\\_epl\) example”](#) on page 1233. AMODE 64 callers use the [“BPX4EPL \(epoll\\_epl\) example”](#) on page 1327. For the data structure, see [“BPXYEPOL— Map epoll syscall parameters”](#) on page 1075.

```

XC      EPOLLEVENT(EPOLLEV#LEN),EPOLLEVENT
MVC     EPOLL_ID(8),=CL8'EPOLLEV '
MVI     EPOLLEVENTS,EPOLLIN+EPOLLOUT
MVC     EPOLL_FD(4),FILE_FD
CALL    BPX1EPL,          Control an epoll file
      (EPOLL_FD,          Input: Epoll instance from BPX1EPC
      EPOLL_CTL_ADD,      Input: Operation
      FILE_FD,            Input: file descriptor to monitor
      EPOLLEV#LEN,        Input: EpollEvent length
      EPOLLEVENT,         Input: EpollEvent Area
      RETVAL,             Return Value: 0 or -1
      RETCODE,            Return code
      RSNCODE),           Reason code
      VL,MF=(E,PLIST)

```

## BPX1EPW (epoll\_wait) example

The following code creates an **epoll\_wait** instance. For the callable service, use “epoll\_wait (BPX1EPW, BPX4EPW) — Wait for an I/O event on an epoll file descriptor” on page 157. AMODE 64 callers use the “BPX4EPW (epoll\_wait) example” on page 1327. For the data structure, see “BPXYEPOL— Map epoll syscall parameters” on page 1075.

```

LA      R15,BUFFERA
USING   EPOLLEVENT,R15
CALL    BPX1EPW,      Wait for I/O events
          (EPOLL_FD,    Input: Epoll instance from BPX1EPC
          MAXEVENTS,    Input: Max events fitting in EPOLLEVENT buffer
          TIMEOUT,      Input: How long to wait
          BUFFLEN,      Input: Length of buffer
          EPOLLEVENT,   Input: Epoll Events buffer
          RETVAL,       Return Value: -1 or Num Events returned
          RETCODE,      Return code
          RSNCODE),     Reason code
          VL,MF=(E,PLIST)

```

## BPX1EXC (exec) example

The program ictasma located at ict/bin gets control and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see “exec (BPX1EXC, BPX4EXC) — Run a program” on page 163. AMODE 64 callers use “BPX4EXC (exec) example” on page 1327.

```

MVC     BUFLINA,=F'16'
MVC     BUFFERA(16),=C'/ict/bin/ictasma'
MVC     ARGCNT,=F'3'

*                               First
LA      R15,=F'4'              Length
ST      R15,ARGLIST+00         Length parm list
LA      R15,=CL4'WK18'         Argument
ST      R15,ARGSLIST+00        Argument address parm list

*                               Second
LA      R15,=F'7'              Length
ST      R15,ARGLIST+04         Length parm list
LA      R15,=CL7'DEPT37A'      Argument
ST      R15,ARGSLIST+04        Argument address parm list

*                               Third
LA      R15,=F'22'             Length
ST      R15,ARGLIST+08         Length parm list
LA      R15,=CL22'RATE(STD,NOEXC,NOSPEC)' Argument
ST      R15,ARGSLIST+08        Argument address parm list

*                               Number of env. data items passed
MVC     ENVCNT,=F'0'
MVC     ENVLENS,=F'0'          Addr of env. data length list
MVC     ENVPARMS,=F'0'         Add of env. data

*                               ->exit routine
MVC     EXITRTNA,=V(EXITRTN)
MVC     EXITPLA,=A(exit parameter list as expected by EXITRTN)

SPACE   ,
CALL    BPX1EXC,
          (BUFLINA,      Input: Pathname length      +
          BUFFERA,       Input: Pathname              +
          ARGCNT,        Input: Argument count        +
          ARGLIST,       Input: Argument length list   +
          ARGSLIST,      Input: Argument address list  +
          ENVCNT,        Input: Environment count     +
          ENVLENS,       Input: Environment length list +
          ENVPARMS,      Input: Environment address list +
          EXITRTNA,      Input: Exit routine address or 0 +
          EXITPLA,       Input: Exit Parm list address or 0+
          RETVAL,        Return value: -1 or not return +
          RETCODE,       Return code                  +
          RSNCODE),     Reason code                  +
          VL,MF=(E,PLIST) -----

```



## BPX1EXI (\_exit) example

The following code ends the program and returns an exit code of 44 to the waiting parent process. For the callable service, see “\_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup” on page 180. AMODE 64 callers use “BPX4EXI (\_exit) example” on page 1328.

```

XC      WAST(WAST#LENGTH),WAST
MVI     WASTEXITCODE,44      User defined exit code
SPACE
CALL    BPX1EXI,             End a process                +
      (WAST),                Input: Status field          +
      VL,MF=(E,PLIST)        -----

```

## BPX1EXM (execmvs) example

The following code invokes program APPL92 and passes the length and parameter MONTH9,PRELIM, (232/74.99). There is no exit routine associated with program APPL92. For the callable service, see “execmvs (BPX1EXM, BPX4EXM) — Run an MVS program” on page 175. AMODE 64 callers use “BPX4EXM (execmvs) example” on page 1328.

```

MVC     PGMNAMEL,=F'6'
MVC     PGMNAME(06),=CL6'APPL92'
MVC     BUFLINA,=F'24'
MVC     BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
SPACE
CALL    BPX1EXM,             Invoke a MVS program          +
      (PGMNAMEL,             Input: Length of program name  +
      PGMNAME,               Input: Program name           +
      BUFLINA,               Input: Length of program argument +
      BUFFERA,               Input: Program argument        +
      =A(0),                 Input: Exit routine address or 0 +
      =A(0),                 Input: Exit Parm list address or 0+
      RETVAL,                Return value: -1 or not return  +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)        -----

```

## BPX1EXT (extlink\_np) example

The following code creates an external link to data set MY.DATASET for path name /mvs/mydataset. For the callable service, see “extlink\_np (BPX1EXT, BPX4EXT) — Create an external symbolic link” on page 183. AMODE 64 callers use “BPX4EXT (extlink\_np) example” on page 1329.

```

MVC     BUFFERA(10),=CL10'MY.DATASET'
MVC     BUFLINA,=F'10'
MVC     BUFFERB(14),=CL14'/mvs/mydataset'
MVC     BUFLINB,=F'14'
SPACE
CALL    BPX1EXT,             Create external link to name   +
      (BUFLINA,              Input: External name length   +
      BUFFERA,               Input: External name          +
      BUFLINB,               Input: Link name length        +
      BUFFERB,               Input: Link name              +
      RETVAL,                Return value: 0 or -1          +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)        -----

```

## BPX1FAA (faccessat) example

The following code creates the file for the file identified by pathname within the directory represented by the file descriptor FILEFD. Previously, FILEFD would have been to the file descriptor of the desired directory or to AT\_FDCWD to indicate the current directory. For the callable service, see “faccessat

## BPX1FAI (freeaddrinfo) example

(BPX1FAA, BPX4FAA) — Checks user's permissions of a file relative to a directory file descriptor” on page 186. AMODE 64 callers use “BPX4FAA (faccessat) example” on page 1329.

```
MVC  BUFFERA(18),=CL18'/usr/inv/network.t'
MVC  BUFLINA,=F'18'
XC   ACC(ACC#LENGTH),ACC
MVI  ACCINTENTFLAGS,ACC_R_OK+ACC_W_OK  Read and write access
XC   ATFLAGS,ATFlags
MVI  ATFLAGS,AT_EACCESS  Use effective user ID
SPACE ,
CALL  BPX1FAA,                Determine accessibility of a file +
      (BUFLINA,              Input: Pathname length          +
      BUFFERA,               Input: Pathname                +
      ACC,                   Input: Access, BPXYACC           +
      FILEDESC,              Input: Directory file descriptor +
      ATFLAGS,               Input: At flags                  +
      RETVAL,                Return value: 0 or -1            +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      VL,MF=(E,PLIST)        -----
SPACE ,
ICM   R15,B'1111',RETVAL     Set condition code for RETVAL
BZ    PSEUDO                 Branch if RETVAL is zero
CLC   RETCODE,=A(EACCES)     Compare RETCODE to EACCES
BE    PSEUDO                 Branch if access denied
```

## BPX1FAI (freeaddrinfo) example

The following code frees the Addr\_Info structure(s) that were obtained by the getaddrinfo callable service. For the callable service, see “freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr\_Info structures” on page 236. AMODE 64 callers use “BPX4FAI (freeaddrinfo) example” on page 1329.

```
SPACE ,
CALL  BPX1FAI,                Free Addr_Info                +
      (ADDR_INFO_PTR,        Input: -> Addr_Info structure  +
      RETVAL,                Return code                    +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)        -----
```

## BPX1FCA (fchaudit) example

The following code changes the audit for the standard input file to ReadFail, WriteFail and ExecFail. For the callable service, see “fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor” on page 199. For the data structure, see “BPXYAUDT — Map flag values for chaudit and fchaudit” on page 1060. AMODE 64 callers use “BPX4FCA (fchaudit) example” on page 1330.

```
MVI  AUDTREADACCESS,AUDTREADFAIL
MVI  AUDTWRITEACCESS,AUDTWRITEFAIL
MVI  AUDTEXECACCESS,AUDTEXECFAIL
MVI  AUDTRSRV,X'00'
SPACE ,
CALL  BPX1FCA,                Change audit                  +
      (=A(STDIN_FILENO),    Input: File descriptor          +
      AUDT,                  Input: Audit flags, BPXYAUDT    +
      =A(0),                 Input: 0 user, 1 security auditor +
      RETVAL,                Return value: 0 or -1            +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      VL,MF=(E,PLIST)        -----
```

## BPX1FCD (fchdir) example

The following code changes the working directory for the task to the directory identified by FILEDESC. For the callable service, see “fchdir (BPX1FCD, BPX4FCD) – Change the working directory” on page 202. AMODE 64 callers use “BPX4FCD (fchdir) example” on page 1330.

MVC	FILEDESC,..	Directory descriptor from opendir	14
SPACE	,		
CALL	BPX1FCD,	Change working directory	+
	(FILEDESC,	Input: Directory file descriptor	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FCM (fchmod) example

The following code changes the permissions for the standard input file. For the callable service, see “fchmod (BPX1FCM, BPX4FCM) – Change the mode of a file or directory by descriptor” on page 204. For the data structure, see “BPXYMODE – Map the mode constants” on page 1106 and “BPXYFTYP – File type definitions” on page 1078. AMODE 64 callers use “BPX4FCM (fchmod) example” on page 1330.

XC	S_MODE,S_MODE		
MVI	S_MODE2,S_IRUSR	All permissions	
MVI	S_MODE3,S_IRWXU2+S_IRWXG+S_IRWXO		
SPACE	,		
CALL	BPX1FCM,	Change file modes	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	S_MODE,	Input: Mode, BPXYMODE, BPXYFTYP	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FCO (fchown) example

The following code changes the owner and group for the standard input file. For the callable service, see “fchown (BPX1FCO, BPX4FCO) – Change the owner and group of a file or directory by descriptor” on page 210. For the data structure, see “BPXYCONS – Constants used by services” on page 1062. AMODE 64 callers use “BPX4FCO (fchown) example” on page 1330.

MVC	GROUPID,..	Group ID	15
MVC	USERID,..	User ID	15
SPACE	,		
CALL	BPX1FCO,	Change the owner and group of file+	
	(=A(STDIN_FILENO),	Input: File descriptor	+
	USERID,	Input: New user ID for file	+
	GROUPID,	Input: New group ID for file	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FCR (fchattr) example

The following code changes the attributes of the standard input file. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see “fchattr (BPX1FCR, BPX4FCR) – Change the attributes of a file or directory by descriptor” on page 190. For the data structures, see “BPXYATT –

## BPX1FCT (fcntl) example

Map file attributes for `chattr` and `fchattr` on page 1057 and “BPXYMODE — Map the mode constants” on page 1106. AMODE 64 callers use “BPX4FCR (fchattr) example” on page 1331.

```
MVC  ATTID,=CL4'ATT '      Eye Catcher
MVC  ATTVERSION,=AL2(ATT#VER) version
XC   S_MODE,S_MODE        Clear mode
MVI  S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI  S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC  ATTMODE,S_MODE        Move mode data to attribute      +
                                structure
MVC  ATTUID,=F'7'          Specify new UID
MVC  ATTGID,=F'77'         Specify new GID
OI   ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG      +
                                Flag Mode, UID and GID changes
OI   ATTSETFLAGS2,ATTCTIMETOD                  +
                                Set change time to current time
SPACE ,
CALL  BPX1FCR,              Change file attributes          +
      (=A(STDIN_FILENO),    Input: File descriptor        +
      =A(ATT#LENGTH),       Input: BPXYATT length         +
      ATT,                  Input/output: BPXYATT          +
      RETVAL,               Return value: 0 or -1          +
      RETCODE,              Return code                   +
      RSNCODE),             Reason code                   +
      VL,MF=(E,PLIST)       -----
```

## BPX1FCT (fcntl) example

The code for the first example duplicates the standard error file descriptor to a file descriptor greater than or equal to `FILEDES2`.

The code for the second example sets a shared byte range lock. For the callable service, see “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 216. For the data structure, see “BPXYFCTL — Command values and flags for fcntl” on page 1076, “BPXYBRLK — Map byte range lock request for fcntl” on page 1060, and “BPXYOPNF — Map flag values for open” on page 1113. AMODE 64 callers use “BPX4FCT (fcntl) example” on page 1331.

```
* for 2nd parm F_DUPFD, F_DUPFD2      3rd parm file desc no..
* for 2nd parm F_GETFD, F_GETFL       3rd parm 0
* for 2nd parm F_SETFD                3rd parm BPXYFCTL
* for 2nd parm F_GETLK, F_SETLK, F_SETLKW 3rd parm BPXYBRLK
* for 2nd parm F_SETFL                3rd parm BPXYOPNF
SPACE ,
* Example 1 - duplicate file descriptor
MVC  FILEDES2,=F'20'      Get free file descriptor >= 20
SPACE ,
CALL  BPX1FCT,            General purpose file control      +
      (=A(STDERR_FILENO), Input: File descriptor          +
      =A(F_DUPFD),        Input: Action, BPXYFCTL          +
      FILEDES2,           Input: Argument #/0/FCTL/BRLK/OPNF+
      RETVAL,             Return value: 0, -1 or action    +
      RETCODE,            Return code                     +
      RSNCODE),           Reason code                     +
      VL,MF=(E,PLIST)     -----
SPACE ,
* Example 2 - duplicate file descriptor
MVC  FILEDES2,=F'20'      Get next higher file descriptor
LA   R15,BRLK
ST   R15,BRLKA
XC   BRLK(BRLK#LENGTH),BRLK      Null out BRLK
MVC  L_TYPE,=AL2(F_RDLCK)        Lock type = shared
MVC  L_WHENCE,=AL2(SEEK_CUR)      Whence = from current cursor
SPACE ,
CALL  BPX1FCT,            General purpose file control      +
      (=A(STDERR_FILENO), Input: File descriptor          +
      =A(F_SETLK),        Input: Action, BPXYFCTL          +
      BRLKA,              Input: Argument #/0/FCTL/BRLK/OPNF+
      RETVAL,             Return value: 0, -1 or action    +
      RETCODE,            Return code                     +
      RSNCODE),           Reason code                     +
      VL,MF=(E,PLIST)     -----
```

## BPX1FLK (flock) example

The following code serializes open files in the z/OS UNIX file system. For the callable service, use “flock (BPX1FLK, BPX4FLK) — Apply or remove an advisory lock on an open file” on page 225. AMODE 64 callers use the “BPX4FLK (flock) example” on page 1332.

```
LHI      @08,14
ST       @08,ACTIONCODE
CALL BPX1FLK(testFileFD,ActionCode,Rv,Rc,Rs);
L        @05,RV
CHI      @05,-1
JNE      @RF00101
```

## BPX1FPC (fpathconf) example

The following code obtains the configurable option associated with the pipe buffer. For the callable service, see “fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables” on page 233. For the data structure, see “BPXYPCF — Command values for pathconf and pathconf” on page 1114. AMODE 64 callers use “BPX4FPC (fpathconf) example” on page 1332.

MVC	FILEDESC,..	From opendir	16
SPACE	,		
CALL	BPX1FPC,	Get configurable pathname variable+	
	(FILEDESC,	Input: Directory file descriptor +	
	=A(PC_PIPE_BUF),	Input: Configurables BPXYPCF +	
	RETVAL,	Return value: 0, -1 or variable +	
	RETCODE,	Return code +	
	RSNCODE),	Reason code +	
	VL,MF=(E,PLIST)	-----	

## BPX1FRK (fork) example

The following code forks a new process. The next sequential instruction gets control from both the parent process (RETVAL=child process ID) and from the child process (RETVAL=0). If RETVAL=-1, the fork failed. For the callable service, see “fork (BPX1FRK, BPX4FRK) — Create a new process ” on page 227. AMODE 64 callers use “BPX4FRK (fork) example” on page 1332.

CALL	BPX1FRK,	Create a new process (fork)	+
	(RETVAL,	Return value: -1, 0, child's PID	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FST (fstat) example

The following code gets the file status for the file opened as FILEDESC. For the callable service, see “fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor ” on page 238. For the data structure, see “BPXYSTAT — Map the response structure for stat ” on page 1162. AMODE 64 callers use “BPX4FST (fstat) example” on page 1333.

MVC	FILEDESC,..	File descriptor from open	17
SPACE	,		
CALL	BPX1FST,	Get file status of file descriptor+	
	(FILEDESC,	Input: File descriptor	+
	STATL,	Input: Length of buffer needed	+
	STAT,	Buffer, mapped by BPXYSTAT	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FSA (fstatat) example

The following code gets the file status of the file that was specified by the path BUFFERA in the directory specified by the directory descriptor DIRDESC. For the callable service, see [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor”](#) on page 241. For the data structure, see [“BPXYSTAT — Map the response structure for stat”](#) on page 1162. A mode 64 callers use [“BPX1FSA \(fstatat\) example”](#) on page 1333.

```

MVC   DIRDESC      Directory Descriptor from OPENDIR
MVC   FLAGS,F='0'   Defined in BPXYFCTL
MVC   BUFLINA,F='6'
MVC   BUFFERA(6),=CL6'file.t'
SPACE
CALL  BPX1FST  Get File Status
      (BUFLINA,   Input: Path length
       BUFFERA,   Input: Path
       STATL,     Stat length
       STAT,      Stat buffer
       DIRDESC,   Input: Directory descriptor
       Flags,     Input: Flags
       RETVAL     Return value: 0 or -1
       RETCODE    Return code
       RSNCODE)   Reason code
      VL,MF=(E,PLIST)

```

## BPX1FSY (fsync) example

The following code writes file descriptor changes to permanent storage. For the callable service, see [“fsync \(BPX1FSY, BPX4FSY\) — Write changes to permanent storage”](#) on page 247. A mode 64 callers use [“BPX4FSY \(fsync\) example”](#) on page 1333.

MVC	FILEDESC,..	File descriptor from open	17
	SPACE		
CALL	BPX1FSY,	Write changes to permanent storage+	
	(FILEDESC,	Input: File descriptor	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FTR (ftruncate) example

The following code truncates the file described by FILEDESC after 512 bytes. For the callable service, see [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file”](#) on page 249. A mode 64 callers use [“BPX4FTR \(ftruncate\) example”](#) on page 1334.

MVC	FILEDESC,..	File descriptor from open	17
MVC	NEWLEN(8),=FL8'512'		
	SPACE		
CALL	BPX1FTR,	Truncate a file	+
	(FILEDESC,	Input: File descriptor	+
	NEWLEN,	Input: Length to keep	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1FTV (fstatvfs) example

The following code obtains information about the file system containing the file identified by FILEDESC. For the callable service, see [“fstatvfs \(BPX1FTV, BPX4FTV\) — Get the file system status”](#) on page 244.

For the data structure, see “BPXYSSTF — Map response structure for file system status” on page 1161. AMODE 64 callers use “BPX4FTV (fstatvfs) example” on page 1334.

MVC	FILEDESC,..	File descriptor from open	18
SPACE	,		
CALL	BPX1FTV,	Get file system status	+
	(FILEDESC,	Input: File descriptor	+
	SSTFL,	Input: Length of BPXYSSTF	+
	SSTF,	Buffer, BPXYSSTF	+
	RETVAL,	Return value: -1 or length status	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1GAI (getaddrinfo) example

The following code returns the IP address and other associated information for the specified node name. For the callable service, see “getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location” on page 251. AMODE 64 callers use “BPX4GAI (getaddrinfo) example ” on page 1334.

SPACE	,		
CALL	BPX1GAI,	Get Addr_info	+
	(NODE_NAME,	Input: Name of Host being queried	+
	NODE_NAME_LENGTH,	Input: Length of host name	+
	SERVICE_NAME,	Input: Service name being queried	+
	SERVICE_NAME_LENGTH,	Input: Length of service name	+
	HINTS_PTR,	Input: Ptr to Addr_Info Structure	+
	RESULTS_PTR,	Output:Ptr to Addr_Info Structure	+
	CANONICAL_LENGTH,	Output: Length canonical name	+
	RETVAL,	Return code	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1GCL (getclientid) example

The following code obtains the clientid information for caller. This information is used on givesocket (BPX1GIV) and takesocket (BPX1TAK) services. For the callable service, see “getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier” on page 258. For the data structure, see “BPXYCID — Map the returning structure for getclientid()” on page 1061. AMODE 64 callers use “BPX4GCL (getclientid) example” on page 1335.

CALL	BPX1GCL,	get clientid information	+
	(=F'2',	Input: Function code of 2	+
	=A(AF_INET),	Input: Domain of AF_INET	+
	CID,	Output: Clientid information	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1GCW (getcwd) example

The following code gets the working directory for the caller. For the callable service, see “getcwd (BPX1GCW, BPX4GCW) — Get the path name of the working directory” on page 260. AMODE 64 callers use “BPX4GCW (getcwd) example ” on page 1335.

MVC	BUFLNA,=F'1024'	Max directory name return area	
SPACE	,		
CALL	BPX1GCW,	Get working directory name	+
	(BUFLNA,	Input: Length directory work area	+
	BUFFERA,	Buffer	+
	RETVAL,	Return value: 0 or -1	+

## BPX1GEG (getegid) example

RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1GEG (getegid) example

The following code gets the effective group ID of the caller. For the callable service, see [“getegid \(BPX1GEG, BPX4GEG\) – Get the effective group ID” on page 263](#). AMODE 64 callers use [“BPX4GEG \(getegid\) example” on page 1335](#).

CALL	BPX1GEG,	Get the effective group ID	+
	(RETVAL),	Return value: effective group ID	+
	VL,MF=(E,PLIST)	-----	

## BPX1GEP (getpgid) example

The following code returns the process group ID for the process identified by the input process ID. The process ID value is set to 1. For the callable service, see [“getpgid \(BPX1GEP, BPX4GEP\) – Get the process group ID” on page 295](#). AMODE 64 callers use [“BPX4GEP \(getpgid\) example” on page 1335](#).

MVC	PROCID,=XL4'00000001'	Value of process ID	
SPACE	,		
CALL	BPX1GEP,	Get process group ID	+
	(PROCID,	Input: Process ID	+
	RETVAL,	Return value: process group ID	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX1GES (getsid) example

The following code returns the process group ID for the session leader of the process identified by the input process ID. The process ID value is set to 1. For the callable service, see [“getsid \(BPX1GES, BPX4GES\) – Get the process group ID of the session leader” on page 317](#). AMODE 64 callers use [“BPX4GES \(getsid\) example” on page 1336](#).

MVC	PROCID,=XL4'00000000'	Value of process ID	
SPACE	,		
CALL	BPX1GES,	Get group ID of session leader	+
	(PROCID,	Input: Process ID	+
	RETVAL,	Return value: process group ID	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX1GET (w\_getipc) example

The following code retrieves information on the first semaphore defined to the system to which the caller has read access. For the callable service, see [“w\\_getipc \(BPX1GET, BPX4GET\) – Query interprocess communications” on page 991](#). For the data structure, see [“BPXYIPCQ – Map w\\_getipc structure” on page 1098](#). AMODE 64 callers use [“BPX4GET \(w\\_getipc\) example” on page 1336](#).

XC	TOKEN,TOKEN	Zero, token for 1st member
LA	R5,BUFFERA	Area for query IPC return data
ST	R5,BUFA	R5 -> IPCQ
SPACE	,	



CALL	BPX1GET,	Interprocess Communications	+
	(TOKEN,	Input: member token	+
	BUFA,	Input: ->IPCQ	BPXYIPCQ+
	=A(IPCQ#LENGTH),	Input: Length of IPCQ	BPXYIPCQ+
	=A(IPCQ#SEM),	Input: Request	BPXYIPCQ+
	RETVL,	Return value: 0, -1 or value	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
SPACE	,		
L	R15,RETVL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	
LTR	R15,R15	Test for 0 return	
BZ	PSEUDO	Branch on end of file	
ST	R15,TOKEN	Save token for next w_semicp	

## BPX1GEU (geteuid) example

The following code gets the effective user ID of the caller. For the callable service, see [“geteuid \(BPX1GEU, BPX4GEU\) — Get the effective user ID” on page 264](#). AMODE 64 callers use [“BPX4GEU \(geteuid\) example” on page 1336](#).

CALL	BPX1GEU,	Get the effective user ID	+
	(RETVL),	Return value: effective user ID	+
	VL,MF=(E,PLIST)	-----	

## BPX1GGE (getgrent) example

The following code accesses the group database starting with the next available entry and continuing until end of file on the database. It returns a structure identifying information about each group entry in the database. For the callable service, see [“getgrent \(BPX1GGE, BPX4GGE\) — Sequentially access the group database” on page 266](#). For the data structure, see [“BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 1080](#). AMODE 64 callers use [“BPX4GGE \(getgrent\) example” on page 1336](#).

GGELOOP	DS	0H	
	CALL	BPX1GGE,	Access the group database
		(RETVL,	Return value: 0 or ->BPXYGIDS
		RETCODE,	Return code
		RSNCODE),	Reason code
		VL,MF=(E,PLIST)	-----
	ICM	R8,B'1111',RETVL	
	BZ	CHKGGERR	Error or end of file
	USING	GIDS,R8	
*		access the group structure	
	DROP	R8	
	B	GGELOOP	Check next group entry
CHKGGERR	DS	0H	
	ICM	R8,B'1111',RETCODE	
	BZ	GGEEOF	End of file
*		handle error as needed	
GGEEOF	DS	0H	

## BPX1GGI (getgrgid) example

The following code accesses the group database by the ID of the caller and returns a structure identifying the groups by ID. The group ID value is set to 5. For the callable service, see [“getgrgid \(BPX1GGI, BPX4GGI\) — Access the group database by ID” on page 268](#). For the data structure, see [“BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 1080](#). AMODE 64 callers use [“BPX4GGI \(getgrgid\) example” on page 1337](#).

MVC	GROUPID,=XL4'00000005'	Value of group ID	
SPACE	,		
CALL	BPX1GGI,	Access the group database	+
	(GROUPID,	Input: Group ID	+
	RETVL,	Return value: 0 or ->BPXYGIDS	+

## BPX1GGN (getgrnam) example

```

                                RETCODE,          Return code          +
                                RSNCODE),          Reason code          +
                                VL,MF=(E,PLIST)    -----
ICM  R8,B'1111',RETVAl
BZ   NOGIDS
USING GIDS,R8
*    access the group structure
DROP R8
NOGIDS EQU *
```

## BPX1GGN (getgrnam) example

The following code accesses the group database by the name of the caller and returns a structure identifying the groups by ID. For the callable service, see [“getgrnam \(BPX1GGN, BPX4GGN\) — Access the group database by name”](#) on page 271. For the data structure, see [“BPXYGIDS — Map data returned for getgrnam and getgrpid”](#) on page 1080. AMODE 64 callers use [“BPX4GGN \(getgrnam\) example”](#) on page 1337.

```

MVC  GRNAMELN,=F'7'
MVC  GRPGMNAME(7),=CL7'EXTSERV'
SPACE ,
CALL  BPX1GGN,          Access the group database          +
      (GRNAMELN,        Input: Length of group name        +
      GRPGMNAME,        Input: Name of group                +
      RETVAL,           Return value: 0 or ->BPXYGIDS        +
      RETCODE,          Return code                          +
      RSNCODE),         Reason code                          +
      VL,MF=(E,PLIST)   -----
```

## BPX1GGR (getgroups) example

The following code provides the caller with a list of supplementary group IDs. The code sets BUFW size to 256. The actual BUFW size is determined from the previous BPX1GGR RETVAL when BUFW was 0. For the callable service, see [“getgroups \(BPX1GGR, BPX4GGR\) — Get a list of supplementary group IDs”](#) on page 274. AMODE 64 callers use [“BPX4GGR \(getgroups\) example”](#) on page 1338.

```

*    MVC  BUFW,=XL4'00000256'  Value of buffer BUFW
LA     R15,BUFFERA             Space for BUFW words
ST     R15,BUFA                ->Array for group IDs
SPACE ,
CALL  BPX1GGR,                Get list of supplementary grp IDs +
      (BUFW,                  Input: Group ID list size        +
      BUFA,                   ->Buffer for Group ID list address+
      RETVAL,                  Return value: -1, 0, ID count    +
      RETCODE,                 Return code                      +
      RSNCODE),                Reason code                      +
      VL,MF=(E,PLIST)         -----
```

## BPX1GHA (gethostbyaddr) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose address is specified as input. For the callable service, see [“gethostbyaddr \(BPX1GHA, BPX4GHA\) Get the IP address and alias of a host name for the specified IP address”](#) on page 279. AMODE 64 callers use [“BPX4GHA \(gethostbyaddr\) example”](#) on page 1338.

The HOSTENT structure has the following format:

- h\_name - The address of the host name returned by the service. The host name is a variable length field that is ended by x'00'. # \$
- h\_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h\_addrtype - The value 2, which signifies AF\_INET.
- h\_length - The length of the host internet addresses pointed to by h\_addr\_list.

- h\_addr\_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

```

*      MVC  HOST_ADDR,=XL4'C90E0256'   IP Address of Host
      MVC  HOST_ADDRLen,=F'4'         Address length
      SPACE ,
      CALL  BPX1GHA,                  Get host by address      +
          (HOST_ADDR,                Input: IP address of queried HOST +
          HOST_ADDRLen,              Input: Length of IP address      +
          HOSTENT_PTR,               Output: 0 or -> HOSTENT structure +
          =A(AF_INET),               Input: Domain - AF_INET        +
          RETVAL,                    Return code                  +
          RETCODE,                   Return code                  +
          RSNCODE),                  Reason code                  +
          VL,MF=(E,PLIST)            -----

```

## BPX1GHN (gethostbyname) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose domain name is specified as input. For the callable service, see [“gethostbyname \(BPX1GHN, BPX4GHN\) Get IP information for specified host domain names” on page 282](#). AMODE 64 callers use [“BPX4GHN \(gethostbyname\) example” on page 1338](#).

The HOSTENT structure has the following format:

- h\_name - The address of the host name returned by the service. The host name is a variable length field that is ended by x'00'.
- h\_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h\_addrtype - The value 2, which signifies AF\_INET.
- h\_length - The length of the host internet addresses pointed to by h\_addr\_list.
- h\_addr\_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

```

      MVC  HOST_NAME(8),=CL8'HOST1234'
      MVC  HOST_NAMELEN,=F'8'
      SPACE ,
      CALL  BPX1GHN,                  Get host by name      +
          (HOST_NAME,                Input: Name of Host being queried +
          HOST_NAMELEN,              Input: Length of host name      +
          HOSTENT_PTR,               Output: 0 or -> HOSTENT structure +
          RETVAL,                    Return code                  +
          RETCODE,                   Return code                  +
          RSNCODE),                  Reason code                  +
          VL,MF=(E,PLIST)            -----

```

## BPX1GID (getgid) example

The following code gets the real group ID of the caller. For the callable service, see [“getgid \(BPX1GID, BPX4GID\) — Get the real group ID” on page 265](#). AMODE 64 callers use [“BPX4GID \(getgid\) example” on page 1339](#).

```

      CALL  BPX1GID,                  Get the real group ID      +
          (RETVAL),                  Return value: real group ID +
          VL,MF=(E,PLIST)            -----

```

## BPX1GIV (givesocket) example

The following code gives a socket to the program identified by CID (clientid). The target program may then use takesocket (BPX1TAK) to take the socket. SOCKDESC was previously set by a call to BPX1ACP. CID is set by the getclientid (BPX1GCL) service. For the callable service, see [“givesocket \(BPX1GIV,](#)

## BPX1GLG (getlogin) example

BPX4GIV) — Give a socket to another program ” on page 332. For the data structure, see “BPXYCID — Map the returning structure for getclientid()” on page 1061. AMODE 64 callers use “BPX4GIV (givesocket) example” on page 1339.

```
CALL BPX1GIV,          give a socket to another program +
      (SOCKDESC,       Input: Socket descriptor         +
      CID,             Input: Clientid of recipient      +
      RETVAL,          Return value: 0 or -1             +
      RETCODE,         Return code                     +
      RSNCODE),        Reason code                     +
      VL,MF=(E,PLIST)  -----
```

## BPX1GLG (getlogin) example

The following code gets the login name of the caller. For the callable service, see “getlogin (BPX1GLG, BPX4GLG) — Get the user login name” on page 289. AMODE 64 callers use “BPX4GLG (getlogin) example” on page 1339.

```
CALL BPX1GLG,          Get the login name               +
      (RETVAL),        Returns value, 0 or ->login name  +
      VL,MF=(E,PLIST)  -----
```

## BPX1GMN (w\_getmntent) example

The following code gets the mount entries for the caller. For the callable service, see “w\_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems ” on page 995. For the data structure, see “BPXYMNTE — Map response and element structure of w\_getmntent” on page 1104. AMODE 64 callers use “BPX4GMN (w\_getmntent) example” on page 1340.

If BPXYMNTE is assembled with MNTE2=YES, fields MNTEHID and MNTEHLEN must be initialized.

```
LA    R14,MNTEH          R14->MNTEH and MNTE
L     R15,MNTEL           R15 = Length of MNTEH and MNTE
XR    R0,R0              Dummy 2nd operand
XR    R1,R1              Pad=null, length=0
MVCL  R14,R0             Null out MNTEH and MNTE
MVC   MNTEHID,=CL4'MNT2'  Version indicator
MVC   MNTEHLEN,=A(MNTE#LENGTH) Length of MNTE
CALL  BPX1GMN,           Get mount entries               +
      (MNTEL,            Input: Length BPXYMNTE + MNTEH  +
      MNTEH,             Header in BPXYMNTH             +
      RETVAL,            Return value: -1 or mount entries +
      RETCODE,           Return code                     +
      RSNCODE),          Reason code                     +
      VL,MF=(E,PLIST)    -----
```

## BPX1GNI (getnameinfo) example

The following code resolves a socket address into a host name and a service name. For the callable service, see “getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address” on page 291. AMODE 64 callers use “BPX4GNI (getnameinfo) example” on page 1340.

```
SPACE ,
CALL  BPX1GNI,           Get name info                   +
      (SOCKADDR,         Input: Socket address          +
      SOCKADDR_LENGTH,   Input: Length of socket address +
      SERVICE_BUFFER,     I/O: Buffer for service name   +
      SERVICE_BUFFER_LENGTH, I/O: Length of service buffer +
      HOST_BUFFER,        I/O: Buffer for host name      +
      HOST_BUFFER_LENGTH, I/O: Length of host buffer    +
      FLAGS,             Input: Flags                   +
      RETVAL,            Return code                     +
      RETCODE,           Return code                     +
      VL,MF=(E,PLIST)    -----
```

RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1GPG (getpgrp) example

The following code gets the process group ID of the caller. For the callable service, see “[getpgrp \(BPX1GPG, BPX4GPG\) – Get the process group ID](#)” on page 297. AMODE 64 callers use “[BPX4GPG \(getpgrp\) example](#)” on page 1340.

CALL BPX1GPG,	Get the process group ID	+
(RETVAL),	Return value: group ID	+
VL,MF=(E,PLIST)	-----	

## BPX1GNM (getpeername or getsockname) example

The following code gets the peer name, and then requests the socket name. SOCKDESC was returned by a previous call to BPX1SOC. For the callable service, see “[getsockname or getpeername \(BPX1GNM, BPX4GNM\) - Get the name of a socket or connected peer](#)” on page 319. For the data structure, see “[BPXYSOCK – Map SOCKADDR structure and constants](#)” on page 1151. AMODE 64 callers use “[BPX4GNM \(getpeername or getsockname\) example](#)” on page 1340.

SPACE ,		
CALL BPX1GNM,	Get peername	+
(SOCKDESC,	Input: Socket Descriptor	+
SOCK#GNMOPTGETPEERNAME,	Input: Indicate getpeername	+
SOCK#LEN+SOCK_SUN#LEN,	Input: Length - Sockaddr	+
SOCKADDR,	Input: Sockaddr structure	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	
SPACE ,		
CALL BPX1GNM,	Get sockname	+
(SOCKDESC,	Input: Socket Descriptor	+
SOCK#GNMOPTGETSOCKNAME,	Input: Indicate getpeername	+
SOCK#LEN+SOCK_SUN#LEN,	Input: Length - Sockaddr	+
SOCKADDR,	Input: Sockaddr structure	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1GPE (getpwent) example

The following code accesses the user database starting with the next available entry and continuing until end of file on the database. It returns a structure identifying information about each user entry in the database. For the callable service, see “[getpwent \(BPX1GPE, BPX4GPE\) – Sequentially access the user database](#)” on page 302. For the data structure, see “[BPXYGIDN – Map data returned for getpwnam and getpwuid](#)” on page 1079. AMODE 64 callers use “[BPX4GPE \(getpwent\) example](#)” on page 1341.

GPELOOP DS 0H	
CALL BPX1GPE,	Access the user database
(RETVAL,	Return value: 0 or ->BPXYGIDN
RETCODE,	Return code
RSNCODE),	Reason code
VL,MF=(E,PLIST)	-----
ICM R8,B'1111',RETVAL	
BZ CHKGPER	Error or end of file
USING GIDN,R8	
* access the user structure	
DROP R8	
B GPELOOP	Check next user entry
CHKGPER DS 0H	
ICM R8,B'1111',RETCODE	
BZ GPPEOF	End of file

## BPX1GPI (getpid) example

```
*      handle error as needed
GPEEOF DS    0H
```

## BPX1GPI (getpid) example

The following code gets the process ID of the caller. For the callable service, see “[getpid \(BPX1GPI, BPX4GPI\) — Get the process ID](#)” on page 298. AMODE 64 callers use “[BPX4GPI \(getpid\) example](#)” on page 1341.

```
CALL  BPX1GPI,          Get the process ID          +
      (RETVAL),         Returns value, Process ID    +
      VL,MF=(E,PLIST)   -----
```

## BPX1GPN (getpwnam) example

The following code accesses the group database by the user ID of the caller and returns a structure identifying the groups by name. For the callable service, see “[getpwnam \(BPX1GPN, BPX4GPN\) — Access the user database by user name](#)” on page 305. For the data structure, see “[BPXYGIDN — Map data returned for getpwnam and getpwuid](#)” on page 1079. AMODE 64 callers use “[BPX4GPN \(getpwnam\) example](#)” on page 1341.

```
MVC   USERNLEN,=F'8'
MVC   USERNAME(8),=CL8'Pebbles'
SPACE
CALL  BPX1GPN,          Access the user database      +
      (USERNLEN,        Input: Length of user name   +
      USERNAME,         Input: Name of user          +
      RETVAL,           Return value 0 or ->BPXYGIDN  +
      RETCODE,          Return code                  +
      RSNCODE),         Reason code                  +
      VL,MF=(E,PLIST)   -----
```

## BPX1GPP (getppid) example

The following code gets the process ID of the caller's parent. For the callable service, see “[getppid \(BPX1GPP, BPX4GPP\) — Get the parent process ID](#)” on page 299. AMODE 64 callers use “[BPX4GPP \(getppid\) example](#)” on page 1342.

```
CALL  BPX1GPP,          Get PID of the parent process +
      (RETVAL),         Returns value, parent's process ID+
      VL,MF=(E,PLIST)   -----
```

## BPX1GPS (w\_getpsent) example

The following code gets process data associated with the first relative process (PROCTOK=0) to which the caller is authorized access (by the security access facility). For the callable service, see “[w\\_getpsent \(BPX1GPS\) — Get process data](#)” on page 999. For the data structure, see “[BPXYPGPS — Map the response structure for w\\_getpsent](#)” on page 1116.

```
LA     R15,PGPS          Getmain area mapped by BPXYPGPS
ST     R15,PGPSA         Hold pointer to this area
XC     PROCTOK,PROCTOK    First relative process (Zero)
LA     R2,PGPSCONTTYBUF   Controlling TTY ->buffer
ST     R2,PGPSCONTTYPTR   Store into PGPS
MVC    PGPSCONTTYBLEN,=A(L'PGPSCONTTYBUF) Length
LA     R2,PGPSPATHBUF     Pathname ->buffer
ST     R2,PGPSPATHPTR     Store into PGPS
MVC    PGPSPATHBLEN,=A(L'PGPSPATHBUF) Length
LA     R2,PGPSCMDBUF      Command ->buffer
ST     R2,PGPSCMDPTR      Store into PGPS
MVC    PGPSCMDBLEN,=A(L'PGPSCMDBUF) Length
```

SPACE ,		
CALL BPX1GPS,	Get process data	+
(PROCTOK,	Input: Relative process token	+
PGPSL,	Input: Length of buffer needed	+
PGPSA,	I/O: ->Buffer, mapped by BPXYPGPS	+
RETVAL,	Return value: -1, 0, next proctok	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	
SPACE ,		
ICM R15,B'1111',RETVAL	Test Return value: 0 or -1	
ST R15,PROCTOK	The next relative process token	
BZ PSEUDO	RETVAL = 0, end of file	
BM PSEUDO	RETVAL < 0, error	
BP PSEUDO	RETVAL > 0, next logical process	

## BPX1GPT (grantpt) example

The following code grants access to the subsidiary pseudoterminal device that is identified by the file descriptor. For the callable service, see [“grantpt \(BPX1GPT, BPX4GPT\) — Grant access to the subsidiary pseudoterminal”](#) on page 336. AMODE 64 callers use [“BPX4GPT \(grantpt\) example”](#) on page 1342.

CALL BPX1GPT,	Grant access to subsidiary pty	+
(MASTER_FD,	Input: File descriptor	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1GPU (getpwuid) example

The following code accesses the group database by the user name of the caller and returns a structure identifying the groups by name. The code sets the user ID value to 1. For the callable service, see [“getpwuid \(BPX1GPU, BPX4GPU\) — Access the user database by user ID ”](#) on page 308. For the data structure, see [“BPXYGIDN — Map data returned for getpwnam and getpwuid”](#) on page 1079. AMODE 64 callers use [“BPX4GPU \(getpwuid\) example”](#) on page 1342.

MVC USERID,..	Value of user ID	27
SPACE ,		
CALL BPX1GPU,	Access database by user ID	+
(USERID,	Input: User ID	+
RETVAL,	Return value 0 or ->BPXYGIDN	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1GPY (getpriority) example

The following code gets the CPU priority based on the input which and who values. The which value used is PRIO\_PROCESS, which indicates to get the priority by process ID. The who value used is 7, indicating to get the priority for process ID 7. For the callable service, see [“getpriority \(BPX1GPY, BPX4GPY\) — Get the scheduling priority of a process ”](#) on page 300. AMODE 64 callers use [“BPX4GPY \(getpriority\) example”](#) on page 1342.

MVC PROCID,=XL4'00000007'	Process ID to get priority for	
SPACE ,		
CALL BPX1GPY,	Get priority value	+
(=A(PRIO_PROCESS),	Input: Get by Process ID	+
PROCID,	Input: PID to get priority for	+
RETVAL,	Return value: Priority of process	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	
L R15,RETVAL	Load return value	

## BPX1GRD (getrandom) example

C	R15,=F'-1'	Test for -1 return
BE	PSEUDO	Branch on error

## BPX1GRD (getrandom) example

The following code creates a **getrandom** instance. For the callable service, use “[getrandom \(BPX1GRD, BPX4GRD\) — Obtain a series of random bytes](#)” on page 310. AMODE 64 callers use the “[BPX4GRD \(getrandom\) example](#)” on page 1343.

```
MVC FLAGS,=F'2' Flags
MVC BUFLN,=F'4' Address length
LA R15,BUFFERA Space for BUFLN words
ST R15,BUFADR Buffer address
CALL BPX1GRD,          Get random +
    (FLAGS,            Input: FLAGS +
     BUFLN,            Input: Buffer Length +
     BUFADR,           Input: Buffer Address +
     RETVAL,           Return value: random value +
     RETCODE,          Return code +
     RSNCODE),         Reason code +
    VL,MF=(E,PLIST) -----
L R15,RETVAL Load return value
C R15,=F'-1' Test for -1 return
```

## BPX1GRL (getrlimit) example

The following code fills in the `rlimit` structure for the calling process based on the input resource value. The resource value is set to `RLIMIT_CPU`. For the callable service, see “[getrlimit \(BPX1GRL, BPX4GRL\) — Get resource limits](#)” on page 313. For the data structure, see “[BPXYRLIM — Map the rlimit, rusage, and timeval structures](#)” on page 1142. AMODE 64 callers use “[BPX4GRL \(getrlimit\) example](#)” on page 1343.

```
MVC RESOURCE,=A(RLIMIT_CPU) Value of resource
SPACE ,
CALL BPX1GRL,          Get resource limits      +
    (RESOURCE,          Input: resource          +
     RLIMIT,            Structure, mapped by BPXYRLIM +
     RETVAL,            Return value: 0 or -1      +
     RETCODE,           Return code              +
     RSNCODE),          Reason code              +
    VL,MF=(E,PLIST) -----
L R15,RETVAL Load return value
C R15,=F'-1' Test for -1 return
BE PSEUDO Branch on error
```

## BPX1GRU (getrusage) example

The following code fills in the `rusage` structure based on the input who value. The who value is set to `RUSAGE_SELF`. For the callable service, see “[getrusage \(BPX1GRU, BPX4GRU\) — Get resource usage](#)” on page 315. For the data structure, see “[BPXYRLIM — Map the rlimit, rusage, and timeval structures](#)” on page 1142. AMODE 64 callers use “[BPX4GRU \(getrusage\) example](#)” on page 1343.

```
MVC WHO,=A(RUSAGE_SELF) Value of who
SPACE ,
CALL BPX1GRU,          Get resource usage      +
    (WHO,              Input: who              +
     RUSAGE,           Structure, mapped by BPXYRLIM +
     RETVAL,           Return value: 0 or -1      +
     RETCODE,          Return code              +
     RSNCODE),         Reason code              +
    VL,MF=(E,PLIST) -----
L R15,RETVAL Load return value
C R15,=F'-1' Test for -1 return
BE PSEUDO Branch on error
```



## BPX1GTH (\_\_getthent) example

The following code retrieves information on the first process accessible to the caller. For the callable service, see “[\\_\\_getthent \(BPX1GTH, BPX4GTH\) — Get thread data](#)” on page 325. For the data structure, see “[BPXYPGTH — Map the \\_\\_getthent input/output structure](#)” on page 1119. AMODE 64 callers use “[BPX4GTH \(\\_\\_getthent\) example](#)” on page 1344.

```

LA      R5,BUFFERB      R5 -> Input parameters
ST      R5,BUFB         ->input buffer
USING   PGTHA,R5        R5 base for PGTHA
XC      PGTHA(PGTHA#LEN),PGTHA  Null Input area
MVI     PGTHAFLAG1,PGTHAPROCESS+PGTHACOMMAND+PGTHATHREAD
MVI     PGTHAACCESSHID,PGTH#LAST  Last thread
LA      R15,BUFFERA     PgthB, Output buffer
ST      R15,BUFA        ->output buffer
DROP    R5
SPACE   ,
CALL    BPX1GTH,        __getthent
        (=A(PGTHA#LEN),  Input: length input parms BPXYPGTH+
        BUFA,            Input: ->input parms BPXYPGTH+
        =A(1024),        Input: length output area BPXYPGTH+
        BUFB,            Input: ->output area BPXYPGTH+
        RETVAL,          Return value: 0, -1 +
        RETCODE,         Return code +
        RSNCODE),        Reason code +
        VL,MF=(E,PLIST) -----

```

## BPX1GTR (getitimer) example

The following code returns the time remaining an alarm, or ITIMER\_REAL as set by setitimer. For the callable service, see “[getitimer \(BPX1GTR, BPX4GTR\) — Get the value of the interval timer](#)” on page 287. For the data structure, see “[BPXYITIM — Map getitimer, setitimer structure](#)” on page 1101. AMODE 64 callers use “[BPX4GTR \(getitimer\) example](#)” on page 1344.

```

LA      R15,ITIM        Output mapping structure
ST      R15,ITIMA       ->structure
CALL    BPX1GTR,        Get process data +
        (=A(ITIMER_REAL), Input: Relative process token +
        ITIMA,          Out: ->Buffer, mapped by BPXYITIM +
        RETVAL,         Return value: -1, 0 +
        RETCODE,        Return code +
        RSNCODE),       Reason code +
        VL,MF=(E,PLIST) -----

```

## BPX1GUG (getgroupsbyname) example

The following code returns the number of supplementary group IDs, up to 9, for user Pebbles. For the callable service, see “[getgroupsbyname \(BPX1GUG, BPX4GUG\) — Get a list of supplementary group IDs by user name](#)” on page 276. AMODE 64 callers use “[BPX4GUG \(getgroupsbyname\) example](#)” on page 1344.

```

MVC     USERNLEN,=F'7'
MVC     USERNAME(07),=CL07'Pebbles'
MVC     BUFLINA,=F'9'
LA      R15,BUFFERA
ST      R15,BUFA
SPACE   ,
CALL    BPX1GUG,        Get list of groups by user name +
        (USERNLEN,      Input: User name length +
        USERNAME,       Input: User name +
        BUFLINA,        Input: Group ID list size +
        BUFA,           Group ID list address +
        RETVAL,         Return value: -1, or # of grp IDs +
        RETCODE,        Return code +
        RSNCODE),       Reason code +
        VL,MF=(E,PLIST) -----

```

## BPX1GUI (getuid) example

The following code gets the invoker's real user ID. For the callable service, see “getuid (BPX1GUI, BPX4GUI) — Get the real user ID ” on page 329. AMODE 64 callers use “BPX4GUI (getuid) example” on page 1345.

CALL	BPX1GUI,	Get the real user ID	+
	(RETVAL),	Return value: real user ID	+
	VL,MF=(E,PLIST)	-----	

## BPX1GWD (getwd) example

The following code gets the working directory for the caller. For the callable service, see “getwd (BPX1GWD, BPX4GWD) — Get the path name of the working directory” on page 330. AMODE 64 callers use “BPX4GWD (getwd) example” on page 1345.

MVC	BUFLINA,=F'1024'	Max directory name return area	
SPACE	,		
CALL	BPX1GWD,	Get working directory name	+
	(BUFLINA,	Input: Length directory work area	+
	BUFFERA,	Buffer	+
	RETVAL,	Return value: length or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1HST (gethostid or gethostname) example

The following code requests the host id and the host name for an AF\_INET domain. For the callable service, see “gethostid or gethostname (BPX1HST, BPX4HST) — Get ID or name information about a socket host” on page 285. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 64 callers use “BPX4HST (gethostid or gethostname) example” on page 1345.

XC	BUFLINA,BUFLINA		
CALL	BPX1HST,	Request host id	+
	(=A(AF_INET),	Input: Domain - AF_INET	+
	BUFLINA,	Input: Length - No buffer - get id+	
	BUFFERA,	Output: (not used with Length=0)	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

MVC	BUFLINA,=A(L'BUFFERA)		
CALL	BPX1HST,	Request host name	+
	(=A(AF_INET),	Input: Domain - AF_INET	+
	BUFLINA,	Input: Length - for output name	+
	BUFFERA,	Output: Buffer for host name	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1IAW (inotify\_add\_watch) example

The following code creates an **inotify\_add\_watch** instance. For the callable service, use “inotify\_add\_watch (BPX1IAW, BPX4IAW) — Add or modify a watch to an initialized inotify instance” on page 338. AMODE 64 callers use the “BPX4IAW (inotify\_add\_watch) example” on page 1346.

```
XC    aOptions(4),aOptions
```

```

MVI iOptions,IN_ALL_EVENTS
CALL BPX1IAW(      Add-Watch a file/directory
inotifyFD,        Inotify Instance FD from BPXUSINI
aPathNnameLen,    Length of the input pathname
iPathName,        PathName: i.e '/inotify/dir1'.
aOptions,         Mask: IN_ALL_EVENTS: BPXYIFY
RetVal,           Return Value Inotify Watch Descriptor or -1
RetCode,          Return Code
RsnCode);         Reason Code
L @05,RetVal       Loading Return Value
CHI @05,-1         Testing RetVal
BE ERROR          Branch if negative (-1=failure)
ST @05,INOTIFYWD  Save Inotify Watch Descriptor

```

## BPX1INI (inotify\_init) example

The following code creates an **inotify\_init** instance. For the callable service, use “[inotify\\_init \(BPX1INI, BPX4INI\) — Initialize an inotify instance](#)” on page 341. AMODE 64 callers use the “[BPX4INI \(inotify\\_init\) example](#)” on page 1346.

```

XC iOptions(4)iOptions
MVI iOptions,IN_NONBLOCK
CALL BPX1INI(      Create an inotify instance FD      +
iOptions,         Inotify Instance Option: BPXYIFY   +
RETVAL,           Return value Inotify FD or -1      +
RETCODE,          Return code                       +
RSNCODE),         Reason code                      +
L @05,RetVal       Loading return value
CHI @05,-1         Testing RetVal
BE ERROR          Branch if negative (-1 = failure)
ST @05,INOTIFYFD  Save Inotify File Descriptor

```

## BPX1IOC (w\_ioctl) example

The following code conveys a command to the standard output device. To run properly this example needs a command defined by the user for the COMMAND parameter. This command must be understood by the device driver providing support for the output device. For the callable service, see “[w\\_ioctl \(BPX1IOC, BPX4IOC\) — Control I/O](#)” on page 1003. AMODE 64 callers use “[BPX4IOC \(w\\_ioctl\) example](#)” on page 1347.

```

MVC BUFLINA,=F'1024'
MVC COMMAND,=F'123'      User defined command
SPACE ,
CALL BPX1IOC,            I/O Control                +
(=A(STDOUT_FILENO),      Input: File descriptor      +
COMMAND,                 Input: Command              +
BUFLINA,                 Input: Argument length      +
BUFFERA,                 Argument buffer name        +
RETVAL,                  Return value: 0 or -1       +
RETCODE,                  Return code                 +
RSNCODE),                Reason code                 +
VL,MF=(E,PLIST)         -----

```

## BPX1IPT (mvsiptaffinity) example

The following code executes the assembler routine EXITRTN on the IPT of the requesting thread, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see “[mvsiptaffinity \(BPX1IPT, BPX4IPT\) — Run a program on the IPT thread](#)” on page 476. AMODE 64 callers use “[BPX4IPT \(mvsiptaffinity\) example](#)” on page 1347.

```

* MVC EXITRTNA,=V(EXITRTN)  ->Routine address
MVC EXITPLA,=A(EXITPARM)   ->Input parameter list
SPACE ,
CALL BPX1IPT,              +
(EXITRTNA,                 Input: Routine address   +
EXITPLA,                   Input: Parm list address or 0 +

```

## BPX1IRM (inotify\_rm\_watch) example

RETVAL,	Return value: -1 or not return	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1IRM (inotify\_rm\_watch) example

The following code creates an **inotify\_rm\_watch** instance. For the callable service, use “inotify\_rm\_watch (BPX1IRW, BPX4IRW) — Remove existing watches from inotify instances” on page 344. AMODE 64 callers use the “BPX4IRM (inotify\_rm\_watch) example” on page 1346.

```
CALL BPX1IRW(          Remove-Watch of a file/directory
  inotifyFD,          Inotify Instance FD from BPXUSINI
  inotifyWD,          Inotify Watch-Descriptor from BPXUSIAW
  RetVal,             Return Value 0 or -1
  RetCode,            Return Code
  RsnCode);           Reason Code
L      @05,RetVal      Loading Return Value
CHI    @05,-1          Testing RetVal
BE     ERROR           Branch if negative (-1=failure)
```

## BPX1ITY (isatty) example

The following code determines if the standard output device is a terminal. For the callable service, see “isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal” on page 357.

CALL	BPX1ITY,	Determine if device is a TTY	+
	(=A(STDOUT_FILENO),	Input: File descriptor	+
	RETVAL),	Return value: 0 isn't, 1 is	+
	VL,MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BZ	PSEUDO	RETVAL=0 means device not terminal	

## BPX2ITY (isatty) example

The following code determines if the standard output device is a terminal. For the callable service, see “isatty (BPX2ITY, BPX4ITY) (X/Open Version) — Determine whether a file descriptor represents a terminal” on page 359. AMODE 64 callers use “BPX4ITY (isatty) example” on page 1347.

CALL	BPX2ITY,	Determine if device is a TTY	+
	(=A(STDOUT_FILENO),	Input: File descriptor	+
	RETVAL,	Return value: 0 isn't, 1 is, -1	+
	RETCODE,	Return code: describes why VAL=-1	+
	RSNCODE),	Reason code: qualifier on RETCODE	+
	VL,MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BZ	PSEUDO	RETVAL=0 means device not terminal	

## BPX1KIL (kill) example

The following code sends a signal (SIGUSR1) to all processes for which access is allowed in the invoker's process group. For the callable service, see “kill (BPX1KIL, BPX4KIL) — Send a signal to a process ” on page 360. For the data structure, see “BPXYSIGH — Signal constants” on page 1148. AMODE 64 callers use “BPX4KIL (kill) example” on page 1347.

MVC	PROCID,=A(0)	Invoker's process group	
CALL	BPX1KIL,	Send a signal to a process	+
	(PROCID,	Input: Process ID	+
	=A(SIGUSR1#),	Input: Signal	BPXYSIGH +
	=A(0),	Input: Signal options	+
	RETVAL,	Return value: 0 or -1	+

RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1LCO (lchown) example

The following code changes the owner of symbolic link /somedir/somesymlink.c from the current owner to that specified by USERID and GROUPID. For the callable service, see [“lchown \(BPX1LCO, BPX4LCO\) — Change the owner or group of a file, directory, or symbolic link” on page 380](#). AMODE 64 callers use [“BPX4LCO \(lchown\) example” on page 1348](#).

MVC	BUFFERA(22),=CL22'/somedir/somesymlink.c'		
MVC	BUFLINA,=F'22'		
MVC	USERID,..	New owner UID from stat	33
MVC	GROUPID,..	New owner GID from stat	33
SPACE	,		
CALL	BPX1LCO,	Change owner and group of a file	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	USERID,	Input: New owner UID	+
	GROUPID,	Input: New owner GID	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1LCR (lchattr) example

The following code changes the attributes of symbolic link /somedir/somesymlink.c. The security label is set and the file change time is set. For the callable service, see [“lchattr \(BPX1LCR, BPX4LCR\) — Change the attributes of a file or directory or symbolic link” on page 371](#). For the data structures, see [“BPXYATT — Map file attributes for chattr and fchattr” on page 1057](#). AMODE 64 callers use [“BPX4LCR \(lchattr\) example” on page 1348](#).

MVC	BUFFERA(22),=CL22'/somedir/somesymlink.c'		
MVC	BUFLINA,=F'22'		
XC	ATT,ATT	Clear ATT	
MVC	ATTID,=CL4'ATT '	Eye Catcher	
MVC	ATTVERSION,=AL2(ATT#VER)	version	
MVC	ATTSECLABEL,=CL08'SYSMULTI'		
OI	ATTSETFLAGS3,ATTSECLABELCHG		+
	Flag Seclabel update		
OI	ATTSETFLAGS2,ATTCTIMETOD		+
	Set change time to current time		
SPACE	,		
CALL	BPX1LCR,	Change file attributes	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	=A(ATT#LENGTH),	Input: BPXYATT length	+
	ATT,	Input/output: BPXYATT	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1LDX (loadhfs extended) example

The following is an example specifying the Lod\_Directed option. For an example of BPX1LDX/BPX4LDX without the Lod\_Directed option flag specified, see [“BPX1LOD \(loadhfs\) example” on page 1257](#), substituting BPX1LDX/BPX4LDX for BPX1LOD/BPX4LOD. The program ictasma located at ict/bin is loaded into storage and then branched to. Then the CSVDYLPAservice is called to provide serviceability information to the system. The loaded module can then be branched to. When the load module is no longer needed, the serviceability information should be deleted and the module's storage released. For

## BPX1LDX (loadhfs extended) example

the callable service, see “loadhfs extended (BPX1LDX, BPX4LDX) — Direct the loading of an executable into storage” on page 399. AMODE 64 callers use “BPX4LDX (loadhfs extended) example” on page 1348.

```
MVC  BUFLINA,=F'13'
MVC  BUFFERA(16),=C'/ict/bin/ictasma'
MVC  OPTIONS,=AL4(LOD_DIRECTED)  Directed loadhfs to common
OI   OPTIONS+3,X'F1'             Subpool 241
MVC  LIBPTHLN,=A(0)
SPACE ,
CALL  BPX1LDX,  Load program          +
      (BUFLINA,  Input: Pathname length  +
      BUFFERA,   Input: Pathname         +
      OPTIONS,   Input: Options          +
      LIBPTHLN,  Input: Library Path Length  +
      LIBPATH,   Input: Library Path      +
      RTNPARM@,  Return value: -1 or direct load ret parms  +
      RETCODE,   Return code              +
      RSNCODE),  Reason code              +
      MF=(E,PLIST) -----
SPACE ,
L     R15,RTNPARM@          Load return value
C     R15,=F'-1'           Test for -1 return
      BE  PSEUDO            Branch on error
      L     R5,RTNPARM@
MVC  LOCALPARMS(24),0(R5)  Local copy of returned parameters
*
*          Provide serviceability information to system
*
LA    R4,LOCALPARMS
USING DIRECTEDLOADRETURNEDPARMS,R4
L     R5,DIRECTEDLOADMODULEENTRYPT
L     R6,DIRECTEDLOADMODULESTART
L     R7,DIRECTEDLOADMODULELENGTH
XC    LPMEA(LPMEA_LEN),LPMEA
ST     R5,LPMEAENTRYPOINTADDR
ST     R6,LPMEALOADPOINTADDR
ST     R7,LPMEAMODLEN
MVC  LPMEANAME,=C'ICTASMA '
CSVDYLPALPA REQUEST=ADD,          +
      BYADDR=YES,                 +
      MODINFOTYPE=MEMBERLIST,     +
      MODINFO=LPMEA,              +
      NUMMOD=1,                   +
      REQUESTOR=REQID,            +
      RETCODE=RETCODE,            +
      RSNCODE=RSNCODE,            +
      MF=(E,DYLPAL)  Provide serviceability information
L     R15,RETCODE          Load return code
LTR   R15,R15
BNZ   PSEUDO
MVC  LOCALDELTOKEN(8),LPMEADELETETOKEN
SPACE ,
.
.
.
*
*          Call directed loadhfs target module
*
L     R15,DIRECTEDLOADMODULEENTRYPT
BALR  R14,R15  Branch to loaded program
SPACE ,
.
.
.
*
*          When done with directed load hfs module
*          remove serviceability information and
*          release module storage
*
XC    LPMED(LPMED_LEN),LPMED
MVC  LPMEDNAME,=C'ICTASMA '
MVC  LPMEDDELETETOKEN(8),LOCALDELTOKEN
CSVDYLPALPA REQUEST=DELETE,          +
      TYPE=BYTOKEN,                +
      MODINFO=LPMED,               +
      NUMMOD=1,                    +
      RETCODE=RETCODE,              +
      RSNCODE=RSNCODE,              +
      MF=(E,DYLPAL)  Remove serviceability information
```

```

L      R15,RETCODE      Load return code
LTR    R15,R15
BNZ    PSEUDO
SPACE ,
MODESET MODE=SUP
L      R7,DIRECTEDLOADMODULELENGTH
        STORAGE RELEASE,          +
        LENGTH=(R7),              +
        ADDR=DIRECTEDLOADMODULESTART,
        SP=241                    Free module
MODESET MODE=PROB
DROP   R4

```

## BPX1LOD (loadhfs) example

The program ictasma located at ict/bin is loaded into storage and then branched to. For the callable service, see “loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name” on page 393. AMODE 64 callers use “BPX4LOD (loadhfs) example” on page 1350.

```

MVC    BUFLINA,=F'16'
MVC    BUFFERA(16),=C'/ict/bin/ictasma'
MVC    OPTIONS,=A(0)
MVC    LIBPETHLN,=A(0)
SPACE ,
CALL   BPX1LOD,          Load program          +
        (BUFLINA,        Input: Pathname length +
        BUFFERA,         Input: Pathname       +
        OPTIONS,         Input: Options        +
        LIBPETHLN,       Input: Library Path Length +
        LIBPATH,         Input: Library Path   +
        EPADDR,          Return value: -1 or entrypt addr +
        RETCODE,         Return code          +
        RSNCODE),        Reason code          +
        VL,MF=(E,PLIST) -----
SPACE ,
L      R15,EPADDR        Load return value
C      R15,=F'-1'        Test for -1 return
BE     PSEUDO            Branch on error
SPACE ,
L      R15,EPADDR
BALR   R14,R15          Branch to loaded program

```

## BPX1LNA (linkat) example

The following code creates a hard link to an existing file lnk03oldfile01.txt. For the callable service, see “linkat (BPX1LNA, BPX4LNA) — Make a link using at directory relative to a directory file descriptor” on page 387. For the data structure, see “BPXYFCTL — Command values and flags for fcntl” on page 1076. AMODE 64 callers use the “BPX4LNA (linkat) example” on page 1350.

```

CALL   BPX1LNA,          Create a hard link to a file  x
        (=F'18',         Length of existing file name  x
        =C'lnk03oldfile01.txt', Name of exiting file  x
        =F'19',         Length of new path name       x
        =C'lnk03_newfile02.txt', Name of new path name  x
        Olddirfd,       Directory for old file        x
        newdirfd,       Directory for new file        x
        =A(At_SymLink_Follow), Flags                x
        retVal,         x
        my_retCode,     x
        rsnCode),      x
        VL,MF=(E,PLIST)
PLIST  DS      10A
        BPXYFCTL

```

## BPX1LNK (link) example

The following code creates a new way for `usr/dataproc.next.t` to link to an existing file, `usr/user05/yearrecs.t`. For the callable service, see [“link \(BPX1LNK, BPX4LNK\) — Create a link to a file”](#) on page 383. AMODE 64 callers use [“BPX4LNK \(link\) example”](#) on page 1351.

```

MVC  BUFLINA,=F'21'
MVC  BUFFERA(21),=CL21'usr/user05/yearrecs.t'
MVC  BUFLINB,=F'19'
MVC  BUFFERB(19),=CL19'usr/dataproc.next.t'
SPACE ,
CALL  BPX1LNK,          Create a link to a file          +
      (BUFLINA,          Input: Name length: existing    +
      BUFFERA,           Input: Name of existing file    +
      BUFLINB,           Input: Name length: link        +
      BUFFERB,           Input: Name of link to file     +
      RETVAL,            Return value: 0 or -1           +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                    +
      VL,MF=(E,PLIST)    -----

```

## BPX1LSK (lseek) example

The following code changes the file (FILEDESC) offset to 80 bytes past the current offset. For the callable service, see [“lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset”](#) on page 406. For the data structure, see [“BPXYSEEK — Constants for lseek”](#) on page 1144. AMODE 64 callers use [“BPX4LSK \(lseek\) example”](#) on page 1351.

```

MVC  FILEDESC,...      File descriptor from open      34
MVC  OFFSET(08),=FL8'80' Forward 80 Bytes
MVC  REFPT,=A(SEEK_CUR) Current offset of the file
SPACE ,
CALL  BPX1LSK,          Change a file's offset          +
      (FILEDESC,         File descriptor                +
      OFFSET,            I/O: Offset in file            +
      REFPT,             Input: Reference point, BPXYSEEK +
      RETVAL,            Return value: 0 or -1           +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                    +
      VL,MF=(E,PLIST)    -----

```

## BPX1LSN (listen) example

The following code issues a listen on a socket that was previously created and that had a bind done for it. SOCKDESC was returned from the call to BPX1SOC. Set the backlog count to 5. For the callable service, see [“listen \(BPX1LSN, BPX4LSN\) — Prepare a server socket to queue incoming connection requests from clients”](#) on page 391. For the data structure, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151. AMODE 64 callers use [“BPX4LSN \(listen\) example”](#) on page 1351.

```

CALL  BPX1LSN,          Listen on a socket              +
      (SOCKDESC,         Input: Socket Descriptor        +
      =A(5),             Input: Backlog count of 5        +
      RETVAL,            Return value: 0 or -1           +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                    +
      VL,MF=(E,PLIST)    -----

```

## BPX1LST (lstat) example

The following code obtains the file status for the file described by the symbolic name `labrec/sym`. For the callable service, see [“lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link”](#)



by path name” on page 409. For the data structure, see “BPXYSTAT — Map the response structure for stat” on page 1162. AMODE 64 callers use “BPX4LST (lstat) example” on page 1351.

```

* symbolic name established using symlink (BPX1SYM) system call
MVC  BUFFERA(10),=CL10'labrec/sym'
MVC  BUFLINA,=F'10'
SPACE ,
CALL  BPX1LST,          Get file status          +
      (BUFLINA,         Input: Pathname length    +
      BUFFERA,          Input: Pathname          +
      STATL,            Input: Length of buffer needed +
      STAT,             Buffer, mapped by BPXYSTAT  +
      RETVAL,           Return value: 0 or -1      +
      RETCODE,          Return code               +
      RSNCODE),         Reason code               +
      VL,MF=(E,PLIST)  -----

```

## BPX1MAT (shmat) example

The following code attaches a shared memory segment. For the callable service, see “shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment” on page 804. For the data structure, see “BPXYSHM — Map interprocess communication shared memory segments” on page 1147. AMODE 64 callers use “BPX4MAT (shmat) example” on page 1352.

```

CALL  BPX1MAT,          Shared memory segment control +
      (SHM_ID,          Input: Shared memory segment ID +
      SEGADDR,          Input: ST loc for seg address  +
      =A(0),            Input: Flags                  BPXYSHM +
      RETVAL,           Return value: 0, -1 or ->segment +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      VL,MF=(E,PLIST)  -----

```

## BPX1MCT (shmctl) example

The following code retrieves the size of the shared memory segment. For the callable service, see “shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations” on page 809. For the data structure, see “BPXYSHM — Map interprocess communication shared memory segments” on page 1147. AMODE 64 callers use “BPX4MCT (shmctl) example” on page 1352.

```

LA    R15,BUFFERA
ST    R15,BUFA
SPACE ,
CALL  BPX1MCT,          Shared memory segment control +
      (SHM_ID,          Input: Shared memory segment ID +
      =A(IPC_STAT),     Input: Command                  BPXYIPC +
      BUFA,             Input: ->SHMID_DS or 0          BPXYSHM +
      RETVAL,           Return value: 0, -1 or value    +
      RETCODE,          Return code                     +
      RSNCODE),         Reason code                     +
      VL,MF=(E,PLIST)  -----

```

## BPX1MDA (mkdirat) example

The following code creates a new and empty directory pathname of /usr/newprots/ with user read-execute, group write, and other read-execute permissions. For the callable service, see “mkdirat (BPX1MDA, BPX4MDA) — Create a directory relative to a directory file descriptor” on page 423. For the data structure, see “BPXYFTYP — File type definitions” on page 1078 and “BPXYMODE — Map the mode constants” on page 1106. AMODE 64 callers use “BPX4MDA (mkdirat) example” on page 1352.

```

MVC  BUFLINA,=F'7'
MVC  BUFFERA(7),=CL7'testdir'
XC   S_MODE,S_MODE
MVI  S_MODE2,S_IRUSR

```

## BPX1MDT (shmdt) example

```

MVI    S_MODE3,S_IRWXU2+S_IRGRP+S_IROTH
      CALL BPX1MDA      mkdirat
      (BUFLNA,          Input: Path Length
      BUFFERA,          Input: Path Name Address
      S_MODE,           Input: Mode, mapped by BPXYMODE
      FILEDESC,         Input: Directory file descriptor
      RETVAL,           Return value: 0 or -1
      RETCODE,          Return code
      RSNCODE),         Reason code
      VL,MF=(E,PLIST)  -----
L      R15,RETVAL      Load Return Value
C      R15,-F'1'       Test for -1 return value
BE     PSEUDO0E        Branch on Error path
LR     R4,R15          Save return value
```

## BPX1MDT (shmdt) example

The following code detaches a shared memory segment. For the callable service, see “shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment” on page 812. For the data structure, see “BPXYSHM — Map interprocess communication shared memory segments” on page 1147. AMODE 64 callers use “BPX4MDT (shmdt) example” on page 1353.

```

CALL BPX1MDT,          Shared memory segment detach +
      (SEGADDR,         Input: Shared memory segment addr +
      RETVAL,           Return value: 0, -1 or value +
      RETCODE,          Return code +
      RSNCODE),         Reason code +
      VL,MF=(E,PLIST)  -----
```

## BPX1MGT (shmget) example

The following code creates a private shared memory segment of 500 bytes. For the callable service, see “shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment” on page 827. For the data structure, see “BPXYSEM — Map interprocess communication semaphores” on page 1146. AMODE 64 callers use “BPX4MGT (shmget) example” on page 1353.

```

MVC    KEY(4),=A(IPC_PRIVATE) Local to this family
MVI    S_TYPE,IPC_CREAT+IPC_EXCL Must not already exist
MVI    S_MODE1,0              Not used
MVI    S_MODE2,S_IRUSR        All read and write permissions
MVI    S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL   BPX1MGT,               Create a set of semaphores +
      (KEY,                   Input: Shared memory segment KEY +
      =A(500),                Input: Segment size +
      S_MODE,                 Input: Creation flags BPXYIPC +
      RETVAL,                 Return value: -1 or MessageQueue ID +
      RETCODE,                Return code +
      RSNCODE),               Reason code +
      VL,MF=(E,PLIST)        -----
SPACE ,
ICM    R15,B'1111',RETVAL     Test return value
BNP    PSEUDO                 Branch on shmget failure
ST     R15,SHM_ID             Store SHM_ID associated with key
```

## BPX1MKD (mkdir) example

The following code creates a new and empty directory pathname of /usr/newprots/ with user read-execute, group write, and other read-execute permissions. For the callable service, see “mkdir (BPX1MKD, BPX4MKD) — Make a directory” on page 420. For the data structure, see “BPXYFTYP — File type definitions” on page 1078 and “BPXYMODE — Map the mode constants” on page 1106. AMODE 64 callers use “BPX4MKD (mkdir) example” on page 1353.

```

MVC    BUFFERA(14),=CL14' /usr/newprots/'
MVC    BUFLNA,=F'14'
```

```

XC      S_MODE,S_MODE
MVI     S_MODE2,S_IRUSR      Read search write read search
MVI     S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
SPACE   ,
CALL    BPX1MKD,             Make a directory                +
      (BUFLNA,              Input: Pathname length          +
      BUFFERA,              Input: Pathname                  +
      S_MODE,               Input: BPXYMODE and BPXYFTYP      +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       -----

```

## BPX1MKN (mknod) example

The following code creates a FIFO (pipe) named /u/fifos/fifo1 and user read-write, group read, other read permissions. For the callable service, see “mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file” on page 427. For the data structure, see “BPXYFTYP — File type definitions” on page 1078 and “BPXYMODE — Map the mode constants ” on page 1106. AMODE 64 callers use “BPX4MKN (mknod) example” on page 1354.

```

MVC     BUFFERA(14),=CL14'/u/fifos/fifo1'
MVC     BUFLNA,=F'14'
XC      S_MODE,S_MODE
MVI     S_TYPE,FT_FIFO      First in - first out
MVI     S_MODE2,S_IRUSR      Read write read read
MVI     S_MODE3,S_IWUSR+S_IRGRP+S_IROTH
SPACE   ,
CALL    BPX1MKN,            Create FIFO or char special file +
      (BUFLNA,              Input: Pathname length          +
      BUFFERA,              Input: Pathname                  +
      S_MODE,               Input: BPXYMODE and BPXYFTYP      +
      =A(0),                Input: Device id not used here    +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       -----

```

## BPX1MMI (\_\_map\_init) example

The following code creates a shared memory map with 10 map blocks each with a size of 1 meg. For the callable service, see “\_\_map\_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area” on page 412. For the data structure, see “BPXYMMG — Map interface for \_map\_init and \_map\_service ” on page 1101. AMODE 64 callers use “BPX4MMI (\_\_map\_init) example” on page 1354.

```

LA      R2,INITPARM          Set address of init parm list
ST      R2,INITADDR
USING   _MMG_INIT_PARM,R2
XC      _MMG_INIT_PARM(_MMG_INIT_PARM_LEN),_MMG_INIT_PARM
L       R1,=F'10'           Map area to contain 10 blocks
ST      R1,_MMG_NUMBLKS     *
L       R1,=F'1'            Each block is to be 1 meg in size
ST      R1,_MMG_MEGSPERBLK *
SPACE   ,
CALL    BPX1MMI,             +
      (=A(MMG_INIT),        Input: Function code            +
      INITADDR,             Input: __map_init parameter list +
      RETVAL,               Return value: 0, -1             +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       -----

```

## BPX1MMP (mmap) example

The following code changes the protection of a memory mapped area. For the callable service, see “[mmap \(BPX1MMP, BPX4MMP\) — Map pages of memory](#)” on page 435. AMODE 64 callers use “[BPX4MMP \(mmap\) example](#)” on page 1355.

MVC	FILEDESC,...	File descriptor	37
SPACE	,		
CALL	BPX1MMP,	map pages of memory	+
	(MAP_ADDRESS,	Input: address of mapped area	+
	MAP_LENGTH,	Input: area length	+
	=A(MAP_PRIVATE),	Input: Map type	+
	FILEDESC,	Input: File descriptor	+
	FILEOFFSET,	Input: File offset	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1MMS (\_\_map\_service) example

The following code creates three new data blocks within a shared memory map. For the callable service, see “[\\_\\_map\\_service \(BPX1MMS, BPX4MMS\) — Mapped megabyte area services](#)” on page 416. For the data structure, see “[BPXYMMG — Map interface for \\_map\\_init and \\_map\\_service](#)” on page 1101. AMODE 64 callers use “[BPX4MMS \(\\_\\_map\\_service\) example](#)” on page 1355.

LA	R3,SRVCPARM	Set address of init parm list	
ST	R3,SRVCADDR		
USING	_MMG_SERVICE_PARM,R3		
XC	_MMG_SERVICE_PARM(_MMG_SERVICE_PARM_LEN),_MMG_SERVICE_PARM		
LA	R4,MMG_NEWBLOCK	Request that a block be created	
STH	R4,_MMG_SERVICETYPE		
LA	R3,_MMG_SERVICE_PARM_LEN(R3)	Bump to next entry	
STH	R4,_MMG_SERVICETYPE	Create a second block	
LA	R3,_MMG_SERVICE_PARM_LEN(R3)	Bump to next entry	
STH	R4,_MMG_SERVICETYPE	Create the third block	
SPACE	,		
CALL	BPX1MMS,		+
	(=A(MMG_SERVICE),	Input: Function code	+
	SRVCADDR,	Input: __map_service parm list	+
	=F'3',	Input: Three requests to process	+
	_MMG_MAPTOKEN,	Map area token from INIT call	+
	RETVAL,	Return value: 0, -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1MNA (mknodat) example

The following code creates the file for the file identified by Pathname within the directory represented by the file descriptor FILEFD. Prior to this code excerpt, FILEFD would have been to the file descriptor of the desired directory or to AT\_FDCWD to indicate the current directory. For the callable service, see “[mknodat \(BPX1MNA, BPX4MNA\) — Make a directory, a FIFO, a character special, or a regular file relative to a directory file descriptor](#)” on page 431. For the data structure, see “[BPXYFTYP — File type definitions](#)” on page 1078 and “[BPXYMODE — Map the mode constants](#)” on page 1106. AMODE 64 callers use “[BPX4MNA \(mknodat\) example](#)” on page 1355.

MVC	BUFFERA(14),=CL14'/u/fifos/fifo1'	
MVC	BUFLINA,=F'14'	
XC	S_MODE,S_MODE	
MVI	S_TYPE,FT_FIFO	First in - first out
MVI	S_MODE2,S_IRUSR	Read write read read
MVI	S_MODE3,S_IWUSR+S_IRGRP+S_IROTH	
SPACE	,	
CALL	BPX1MNA,	Create FIFO or char special file
	(BUFLINA,	Input: Pathname length

BUFFERA,	Input: Pathname	+
S_MODE,	Input: BPXYMODE and BPXYFTYP	+
=A(0),	Input: Device id not used here	+
FILEDESC	Input: Directory file descriptor	+
RETVAl,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1MNT (mount) example

The following code requests that the file system mount the system file TESTLIB.FILESYS1 and ready it for use. For the callable service, see “mount (BPX1MNT) — Make a file system available ” on page 444. For the data structure, see “BPXYMTM — Map the modes for mount and unmount ” on page 1110.

XCL	MTM(MTM#LENGTH),MTM	
MVI	MTM1,MTMRDWR	Mount mode - read-write
MVC	BUFLINA,=F'2'	Max 1023
MVC	BUFFERA(02),=CL02'/u'	
MVC	FSNAME(44),=CL44'TESTLIB.FILESYS1'	
MVC	FSTYPE(8),=CL08'HFS'	
CALL	BPX1MNT,	Ready a file system for use
	(BUFLINA,	Input: Mount point length
	BUFFERA,	Input: Mount point name
	FSNAME,	Input: File system name (44 char)
	FSTYPE,	Input: File system type (8 char)
	MTM,	Input: Mount mode BPXYMTM
	=A(0),	Input: Parm length, future
	=A(0),	Input: Parm, future
	RETVAl,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX2MNT (\_\_mount) example

The following code requests that the file system \_\_mount the system file and ready it for use. The file system name and mount parameters are encoded into the various fields in the MNTE. See “mount (BPX1MNT) — Make a file system available ” on page 444. AMODE 64 callers use “BPX4MNT (\_\_mount) example” on page 1356.

LA	R14,MNTEH	R14->MNTEH and MNTE
L	R15,MNTEL	R15 = Length of MNTEH and MNTE
XR	R0,R0	Dummy 2nd operand
XR	R1,R1	Pad=null, length=0
MVCL	R14,R0	Null out MNTEH and MNTE
MVC	MNTEHID,=CL4'MNT2'	Version indicator
MVC	MNTEHLEN,=A(MNTE#LENGTH)	Length of MNTE
MVC	MNTENTFSTNAME(08),=CL08'HFS'	HFS type name
MVC	MNTENTFSNAME(44),=CL44'TESTLIB.FILESYS1'	Filesystem
MVC	MNTENTMOUNTPOINT(02),=CL02'/u'	Mount point
MVC	MNTENTPATHLEN,=F'2'	
MVC	MNTENTFSMODE4,=A(MNTENTFSMODERDONLY)	Filesystem mode
CALL	BPX2MNT,	Ready a file system for use
	(MNTEL,	Input: MNTE length (hdr + body)
	MNTEH,	Input: MNTE
	RETVAl,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1MP (mvspause) example

The following code places this thread into an MVS WAIT, to be terminated when a user ECB specified on a prior MVSpauseInit call is POSTed. The MVS WAIT is also terminated if a signal occurs. For the callable

## BPX1MPC (mvspocclp) example

service, see “mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals” on page 479. AMODE 64 callers use “BPX4MP (mvspause) example” on page 1356.

```
CALL BPX1MP,          MVS Pause          +
      (RETVAL,        Return value: 0, -1  +
      RETCODE,        Return code         +
      RSNCODE),       Reason code         +
      VL,MF=(E,PLIST) -----
```

## BPX1MPC (mvspocclp) examples

1. The following code causes all z/OS UNIX-related resources to be released for this thread, and if this is the last thread in the process, for the process.

```
XC      WAST(WAST#LENGTH),WAST
MVI     WASTEXITCODE,57      User defined exit code
SPACE ,
CALL    BPX1MPC,            MVS Process cleanup      +
      (WAST,                Input: Ending status code 0-255  +
      RETVAL,              Return value: 0, -1 or 1      +
      RETCODE,             Return code                 +
      RSNCODE),            Reason code                 +
      VL,MF=(E,PLIST)      -----
```

2. To indicate that the process ended with a specific code, the application should set up the WAST as follows, and then call BPX1MPC:

```
*****
* Set up the WAST (exit status word) *
* with a user defined exit code      *
*****
LA      R3,0                Set R3 with zero and
ST      R3,LOCALWAST        clear the WAST
LA      R14,LOCALWAST       Get address of WAST
MVI     WASTEXITCODE(R14),44 Set exit status
```

3. To indicate that the process ended with a terminating signal, the application should set up the WAST as follows prior to calling BPX1MPC:

```
*****
* Set up the WAST (exit status word) *
* with a terminating signal          *
*****
LA      R3,0                Set R3 with zero and
ST      R3,LOCALWAST        clear the WAST
LA      R14,LOCALWAST       Get address of WAST
MVI     WASTSIGTERM(R14),09 Exit with sigterm (x'09')
```

4. If an application does not care about the terminating status of a process, and the parent will not check the status after issuing a call to the wait service, then the application should set the WAST to zero prior to calling BPX1MPC:

```
*****
* Set up the WAST (exit status word) *
* Do not set any exit codes          *
*****
LA      R3,0                Set R3 with zero and
ST      R3,LOCALWAST        clear the WAST
LA      R14,LOCALWAST       Get address of WAST
```

For the callable service, see “mvspocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources” on page 484. For the data structure, see “BPXYWAST — Map the wait status word” on page 1172. AMODE 64 callers use “BPX4MPC (mvspocclp) example” on page 1356.

## BPX1MPI (mvspauseinit) example

The following code prepares the thread for a subsequent MVSpause invocation. A list of Event Control Block addresses is passed to the system with the last address having the high order bit on. This syscall will use the first ECB pointed to from the list as the signal ECB, therefore at least one ECB address must be passed to the system. For the callable service, see “mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals” on page 479. AMODE 64 callers use “BPX4MPI (mvspauseinit) example” on page 1357.

	LA	R15,BUFFERA	Load address of ECB address list	
	ST	R15,BUFA	Save address for future parameter	
*			to be passed to BPX1MPI	
	SR	R15,R15	Clear R15	
	ST	R15,ECB01	Clear ECB01	
	ST	R15,ECB02	Clear ECB02	
	LA	R15,ECB01	Load address of first ECB	
	ST	R15,BUFFERA	Save ECB address in list of	
*			pointers	
	LA	R15,ECB02	Load address of second ECB	
	ST	R15,BUFFERA+4	Save ECB address in list of	
*			pointers	
	OI	BUFFERA+4,X'80'	Denote end of ECB pointers	
	SPACE	,		
	CALL	BPX1MPI,	MVS Pause initialize	+
		(BUFA,	Input ->list of ECB@, x'80' ended	+
		RETVAL,	Return value: 0, -1	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		VL,MF=(E,PLIST)	-----	

## BPX1MPR (mprotect) example

The following code changes the protection of a memory mapped area. For the callable service, see “mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping” on page 452. AMODE 64 callers use “BPX4MPR (mprotect) example” on page 1357.

	CALL	BPX1MPR,	set protection of a mapped area	+
		(MAP_ADDRESS,	Input: address of mapped area	+
		MAP_LENGTH,	Input: area length	+
		=A(PROT_READ),	Input: Protection options	+
		RETVAL,	Return value: 0 or -1	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		VL,MF=(E,PLIST)	-----	

## BPX1MSD (mvsunsigsetup) example

The following code detaches the invoker from being able to catch signals. For the callable service, see “mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup” on page 495. C AMODE 64 callers use “BPX4MSD (mvsunsigsetup) example” on page 1357.

	CALL	BPX1MSD,	Reregister MVS signals, this task	+
		(SIRTNA,	Signal interface routine address	+
		USERWORD,	User data	+
		INTMASK,	Default override signal set	+
		TERMMASK,	Default terminate signal set	+
		RETVAL,	Return value: 0 or -1	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		VL,MF=(E,PLIST)	-----	

## BPX1MSS (mvssigsetup) example

The following code allows the invoker to catch signals. For the callable service, see “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 487. AMODE 64 callers use “BPX4MSS (mvssigsetup) example” on page 1358.

```
* Each bit of the mask represents a signal 1-64.
MVC  INTMASK(8),=XL8'F000000000000000'      Default sig 1-4
MVC  TERMMASK(8),=XL8'F000000000000000'      Terminate sig 1-4
LA   R15,BUFFERA
ST   R15,USERWORD
SPACE ,
CALL BPX1MSS,                                Register MVS signals, this task +
      (=V(SIRTN),                             Input: Signal interrupt routine +
      USERWORD,                               Input: User data +
      INTMASK,                                Input: Default override signals +
      TERMMASK,                               Input: Default terminate signals +
      RETVAL,                                 Return value: 0 or -1 +
      RETCODE,                                Return code +
      RSNCODE),                               Reason code +
      VL,MF=(E,PLIST)                         -----
```

## BPX1MSY (msync) example

The following code causes the file associated with this mapped area to be updated with the contents of storage. For the callable service, see “msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage” on page 469. AMODE 64 callers use “BPX4MSY (msync) example” on page 1358.

```
MVC  FILEDESC,..                            File descriptor          41
SPACE ,
CALL BPX1MSY,                                synchronize memory with storage +
      (MAP_ADDRESS,                           Input: address of mapped area +
      MAP_LENGTH,                             Input: area length +
      =A(MS_SYNC),                           Input: sync options +
      RETVAL,                                 Return value: 0 or -1 +
      RETCODE,                                Return code +
      RSNCODE),                               Reason code +
      VL,MF=(E,PLIST)                         -----
```

## BPX1MUN (munmap) example

The following code causes a mapped area to be unmapped. For the callable service, see “munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses” on page 473. AMODE 64 callers use “BPX4MUN (munmap) example” on page 1358.

```
CALL BPX1MUN,                                unmap previously mapped addresses +
      (MAP_ADDRESS,                           Input: address of mapped area +
      MAP_LENGTH,                             Input: area length +
      RETVAL,                                 Return value: 0 or -1 +
      RETCODE,                                Return code +
      RSNCODE),                               Reason code +
      VL,MF=(E,PLIST)                         -----
```

## BPX1NIC (nice) example

The following code increases the priority value of the calling process by 1. For the callable service, see “nice (BPX1NIC, BPX4NIC) — Change the nice value of a process” on page 497. AMODE 64 callers use “BPX4NIC (nice) example” on page 1358.

```
MVC  INCR,=F'1'                             Increase priority by 1
SPACE ,
CALL BPX1NIC,                                Change priority value +
      (INCR,                                  Input: Priority change value +
      RETVAL,                                 Return value: new nice value or -1+)
```



	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX1OPD (opendir) example

The following code opens directory /etc/passwd so that it can be read by readdir. For the callable service, see [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory”](#) on page 523. AMODE 64 callers use [“BPX4OPD \(opendir\) example”](#) on page 1359.

MVC	BUFLINA,=F'11'		
MVC	BUFFERA(11),=CL11' /etc/passwd'		
SPACE	,		
CALL	BPX1OPD,	Open a directory	+
	(BUFLINA,	Input: Directory name length	+
	BUFFERA,	Input: Directory name	+
	RETVAL,	Return value:-1 or directory f.d.	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BL	PSEUDO	Branch if negative (-1 = failure)	
ST	R15,DIRECTDES	Store the directory descriptor	

## BPX1OPN (open) example

The following code opens file usr/inv/nov.d with user read/write, group read, and other read. A file descriptor (FILEDESC) is returned. For the callable service, see [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511. For the data structure, see [“BPXYOPNF — Map flag values for open”](#) on page 1113, AMODE 64 callers use [“BPX4OPN \(open\) example”](#) on page 1359. [“BPXYMODE — Map the mode constants”](#) on page 1106, and [“BPXYFTYP — File type definitions”](#) on page 1078.

MVC	BUFFERA(13),=CL13'usr/inv/nov.d'		
MVC	BUFLINA,=F'13'		
XC	S_MODE,S_MODE		
MVI	S_MODE2,S_IRUSR	User read/write, group read,	
MVI	S_MODE3,S_IWUSR+S_IRGRP+S_IROTH	other read	
XC	O_FLAGS(OPNF#LENGTH),O_FLAGS		
MVI	O_FLAGS4,O_CREAT+O_RDWR	Create, open for read and write	
SPACE	,		
CALL	BPX1OPN,	Open a file	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	O_FLAGS,	Input: Access	BPXYOPNF +
	S_MODE,	Input: Mode	BPXYMODE, BPXYFTYP +
	RETVAL,	Return value:-1 or file descriptor	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BL	PSEUDO	Branch if negative (-1 = failure)	
ST	R15,FILEDESC	Store the file descriptor	

## BPX2OPN (openstat) example

The following code opens file usr/inv/nov.d with user read-write, group read and other read, and obtains status about the file. A file descriptor (FILEDESC) is returned. For the callable service, see [“openstat \(BPX2OPN, BPX4OPS\) — Open a file and obtain status information”](#) on page 526. For the data structures, see [“BPXYOPNF — Map flag values for open”](#) on page 1113, [“BPXYSTAT — Map the response structure for stat”](#) on page 1162, [“BPXYMODE — Map the mode constants”](#) on page 1106, and

## BPX1OPT (getsockopt or setsockopt) example

“BPXYFTYP — File type definitions” on page 1078. AMODE 64 callers use “BPX4OPS (openstat) example” on page 1359.

```
MVC  BUFFERA(13),=CL13'usr/inv/nov.d'
MVC  BUFLINA,=F'13'
XC   S_MODE,S_MODE
MVI  S_MODE2,S_IRUSR      User read/write, group read,
MVI  S_MODE3,S_IWUSR+S_IRGRP+S_IROTH  other read
XC   O_FLAGS(OPNF#LENGTH),O_FLAGS
MVI  O_FLAGS4,O_CREAT+O_RDWR Create, open for read and write
SPACE ,
CALL  BPX20PN,            Open a file and get status          +
      (BUFLINA,           Input: Pathname length             +
      BUFFERA,            Input: Pathname                     +
      O_FLAGS,            Input: Access      BPXYOPNF         +
      S_MODE,             Input: Mode      BPXYMODE, BPXYFTYP +
      STATL,              Input: Length of buffer needed      +
      STAT,               Buffer, BPXYSTAT                    +
      RETVAL,             Return value:-1 or file descriptor+
      RETCODE,            Return code                         +
      RSNCODE),           Reason code                         +
      VL,MF=(E,PLIST)     -----
ICM  R15,B'1111',RETVAL   Test RETVAL
BL   PSEUDO               Branch if negative (-1 = failure)
ST   R15,FILEDESC         Store the file descriptor
```

## BPX1OPT (getsockopt or setsockopt) example

The following code gets and then sets socket options. SOCKDESC was returned on a previous call to BPX1SOC. For the callable service, see “getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket” on page 322. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 64 callers use “BPX4OPT (getsockopt or setsockopt) example” on page 1360.

```
MVC  BUFLINA,=A(L'BUFFERA)
CALL  BPX1OPT,            Get socket options                  +
      (SOCKDESC,          Input: Socket Descriptor           +
      =A(SOCK#0PTOPTGETSOCKOPT), Input: Indicate Get socket +
      SOCK#SOCKET,        Input: Level                       +
      SOCK#SO_TYPE,       Input: Option name                 +
      BUFLINA,            Input: Length - option value       +
      BUFFERA,            Input: Option value                +
      RETVAL,             Return value: 0 or -1              +
      RETCODE,            Return code                        +
      RSNCODE),           Reason code                        +
      VL,MF=(E,PLIST)     -----
SPACE ,
MVC  BUFLINA,=A(4)        SO_00BINLINE has length=4
CALL  BPX1OPT,            Set socket options                  +
      (SOCKDESC,          Input: Socket Descriptor           +
      =A(SOCK#0PTOPTSETSOCKOPT), Input: Indicate set socket +
      SOCK#SOCKET,        Input: Level                       +
      SOCK#SO_TYPE,       Input: Option name                 +
      BUFLINA,            Input: Length - option value       +
      SOCK#SO_00BINLINE,  Input: Option value                +
      RETVAL,             Return value: 0 or -1              +
      RETCODE,            Return code                        +
      RSNCODE),           Reason code                        +
      VL,MF=(E,PLIST)     -----
```

## BPX1PAF (\_\_pid\_affinity) example

The following code will add your PID to the target process' affinity list. For the callable service, see “\_\_pid\_affinity (BPX1PAF, BPX4PAF) — Add or delete an entry in a process's affinity list” on page 548. AMODE 64 callers use “BPX4PAF (\_\_pid\_affinity) example” on page 1360.

```
*      MVC  TARPID,....    PID of target
*      MVC  SIGPID,....    PID of this routine
CALL  BPX1PAF,            +
```

(=A(PAF_ADD_PID#),	Function code (add entry)	+
TARPID,	PID of target	+
SIGPID,	PID to receive signal	+
=A(SIGUSR1#),	signal to be generated	+
RETVL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1PAS (pause) example

The following code suspends execution of the invoker's thread until a signal is delivered. For the callable service, see [“pause \(BPX1PAS, BPX4PAS\) — Suspend a process pending a signal”](#) on page 539. AMODE 64 callers use [“BPX4PAS \(pause\) example”](#) on page 1361.

CALL BPX1PAS,	Suspend execution	+
(RETVL,	Return value: -1 or not return	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1PCF (pathconf) example

The following code extracts the current value for the configurable maximum number of bytes in a file name associated with /usr/inv/network.t. For the callable service, see [“pathconf \(BPX1PCF, BPX4PCF\) — Determine configurable path name variables using a path name”](#) on page 535. For the data structure, see [“BPXYPCF — Command values for pathconf and pathconf”](#) on page 1114. AMODE 64 callers use [“BPX4PCF \(pathconf\) example”](#) on page 1361.

MVC BUFFERA(18),=CL18'/usr/inv/network.t'		
MVC BUFLINA,=F'18'		
SPACE ,		
CALL BPX1PCF,	Get configurable pathname variable+	
(BUFLINA,	Input: Pathname length	+
BUFFERA,	Input: Pathname	+
=A(PC_NAME_MAX),	Input: Options BPXYPCF	+
RETVL,	Return value: 0, -1 or variable	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1PCT (pfsctl) example

The following code conveys a command to a Physical File System named ACMEFILE. ACMEFILE doesn't really exist; to actually run this example you would need a real PFS product that supports this function. For the callable service, see [“pfsctl \(BPX1PCT, BPX4PCT\) — Physical file system control”](#) on page 541. AMODE 64 callers use [“BPX4PCT \(pfsctl\) example”](#) on page 1361.

MVC FSTYPE(8),=CL08'ACMEFILE'		
MVC BUFLINA,=F'25'		
MVC BUFFERA(25),=CL25'COMPRESS(ON) CONVERT(OFF)'		
MVC COMMAND,=F'123'	PFS product defined command	
SPACE ,		
CALL BPX1PCT,	PFS Control	+
(FSTYPE,	Input: PFS Type Name	+
COMMAND,	Input: Command	+
BUFLINA,	Input: Argument length	+
BUFFERA,	Input/Output: Argument buffer	+
RETVL,	Return value: product defined	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1PIP (pipe) example

The following code creates a pipe. For the callable service, see “[pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe](#)” on page 551. AMODE 64 callers use “[BPX4PIP \(pipe\) example](#)” on page 1362.

```
CALL BPX1PIP,          Create a pipe          +
    (READFD,          Output: Read file descriptor +
    WRITEFD,          Output: Write file descriptor +
    RETVAL,           Return value: 0 or -1        +
    RETCODE,          Return code                 +
    RSNCODE),         Reason code                 +
    VL,MF=(E,PLIST)   -----
```

## BPX1PI2 (pipe2) example

The following code creates a **pipe2** instance. For the callable service, use “[pipe2 \(BPX1PI2, BPX4PI2\) — Create a new pipe instance](#)” on page 553. AMODE 64 callers use the “[BPX4PI2 \(pipe2\) example](#)” on page 1362.

```
CALL BPX1PI2,          Create a pipe2 +
    (READFD,          Output: Read file descriptor +
    WRITEFD,          Output: Write file descriptor +
    FLAGS,            Input: Flags +
    RETVAL,           Return value: 0 or -1 +
    RETCODE,          Return code +
    RSNCODE),         Reason code +
    VL,MF=(E,PLIST)
```

## BPX1POE (\_\_poe) example

The following code registers a socket (SOCKDESC) as the process scope port of entry. SOCKDESC was returned previously from a call to either BPX1SOC or BPX1ACP. For the callable service, see “[\\_\\_poe\(\) \(BPX1POE, BPX4POE\) — Port of entry information](#)” on page 559. For the data structure, see “[BPXYPOE — Map poe syscall parameters](#)” on page 1123. AMODE 64 callers use “[BPX4POE \(\\_\\_poe\) example](#)” on page 1362.

```
MVC POEOPTIONS,=A(POE#SCOPEPROCESS)
MVC POENTRYTYPE,=A(POE#ENTRYSOCKET)
MVC POENTRYLEN,=A(POE#ENTRYSOCKETLEN)
LA  R15,SOCKDESC
ST  R15,POENTRYPTR
CALL BPX1POE,          Port of Entry registration  +
    (=A(POE#LEN),      Input: Length of poe structure +
    POE,               Input: mapped by BPXYPOE      +
    RETVAL,            Return value: 0 or -1          +
    RETCODE,           Return code                   +
    RSNCODE),          Reason code                   +
    VL,MF=(E,PLIST)   -----
```

## BPX1POL (poll) example

The following code issues a poll. For the callable service, see “[poll \(BPX1POL, BPX4POL\) — Monitor activity on file descriptors and message queues](#)” on page 564. For the data structure, see “[BPXYPOLL — Map poll syscall parameters](#)” on page 1124. AMODE 64 callers use “[BPX4POL \(poll\) example](#)” on page 1362.

```
*      LA  R15,BUFFERA
      USING POLLFD,R15
      ST  R15,BUFA          ->BPXYPOLL structure
      MVC POLLHFD(4),file_descriptor_number2
      MVI POLLEVENTS,0
      MVI POLLEVENTS+1,POLLERDNORM
      A   R15,=A(POLLFD#LENGTH)
```

```

*      MVC  POLLHFD(4),file_descriptor_number1
      MVI   POLLEVENTS,0
      MVI   POLLEVENTS+1,POLLEWRNORM
      SPACE ,
      CALL  BPX1POL,          Create a pipe          +
      (BUFA,                  Input: address of BPXPOLL      +
      =A(2),                  Input: number of BPXPOLL structs +
      =A(0),                  Input: -1, 0, milliseconds      +
      RETVAL,                 Return value: 0 or -1          +
      RETCODE,                Return code                  +
      RSNCODE),               Reason code                  +
      VL,MF=(E,PLIST)        -----

```

## BPX1PRC (prctl) example

The following code creates a **prctl** instance. For the callable service, use “prctl (BPX1PRC, BPX4PRC) — Managing processes and threads” on page 568. AMODE 64 callers use the “BPX4PRC (prctl) example” on page 1363.

1. The following code sets the name of the calling thread to My job thread 1.

```

      MVC   NAME(15),=CL15'My job thread 1'
      MVC   NAMELEN,=F'15'
      SPACE ,
      CALL  BPX1PRC,          Manage processes and threads +
      (=A(PR_SET_NAME),      Input: prctl constant in BPXYCONS +
      NAMELEN,               Input: Thread name length      +
      NAME,                  Input: Thread name            +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      RETVAL,                Return value: 0 or -1          +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)

```

2. The following code retrieves the name of the calling thread.

```

      CALL  BPX1PRC,          Manage processes and threads +
      (=A(PR_GET_NAME),      Input: prctl constant in BPXYCONS +
      NAMELEN,               Output: Thread name length      +
      NAME,                  Output: Thread name            +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      RETVAL,                Return value: 0 or -1          +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)

```

```

      .
      .
      .

```

## BPX1PRL (prlimit) example

The following code sets the address space resource limits to RLIM\_INFINITY for the specified target PID. The new limits are set in the RLIMIT structure and passed as input. This specific syscall invocation shows only the new limit (set function) that was requested and the old limit parameter is omitted. Alternatively, the prlimit invoker can pass in only the previous limit structure to be filled in by the service or pass in both previous and new limits. For the callable service, see “prlimit (BPX1PRL, BPX4PRL) — Get and set resource limits for specified processes” on page 574. For the data structure, see “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142. AMODE 64 callers use “BPX4PRL (prlimit) example” on page 1363.

```

      XC     NEW_RLIM(RLIMIT#LENGTH),NEW_RLIM  Clear New limit
      LA     R5,NEW_RLIM
      USING  RLIMIT,R5

```

## BPX1PSI (pthread\_setintr) example

```
MVC  RLIM_CUR,=A(RLIM_INFINITY) Current limit
MVC  RLIM_MAX,=A(RLIM_INFINITY) Maximum limit
CALL BPX1PRL,
      (TARGPID,          PID of target process
      =A(RLIMIT_AS),     Address space resource
      NEW_RLIM,          New Limit (Set)
      0,                 Old Limit (Get)
      RETVAL,            +
      RETCODE,           +
      RSNCODE),          +
      MF=(E,PLIST)       +
L     R15,RETVAL          Load return value
C     R15,=F'-1'          Test for -1 return
BE    PSEUDO              Branch on error
```

## BPX1PSI (pthread\_setintr) example

The following code sets the interruption type of the calling thread. For the callable service, see [“pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state” on page 610](#). For the data structure, see [“BPXYCONS — Constants used by services” on page 1062](#). AMODE 64 callers use [“BPX4PSI \(pthread\\_setintr\) example” on page 1364](#).

```
CALL BPX1PSI,          Examine and change interrupt state+
      (INTRSTATE,      Input: Interrupt state  BPXYCONS +
      RETVAL,          Return value: 0 or -1
      RETCODE,         Return code
      RSNCODE),        Reason code
      VL,MF=(E,PLIST)  -----
```

## BPX1PST (pthread\_setintrtype) example

The following code sets the interruption type of the calling thread and returns the previous interruption type. For the callable service, see [“pthread\\_setintrtype \(BPX1PST, BPX4PST\) — Examine and change the interrupt type” on page 613](#) “pthread\_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type” on page 613. For the data structure, see [“BPXYCONS — Constants used by services” on page 1062](#). AMODE 64 callers use [“BPX4PST \(pthread\\_setintrtype\) example” on page 1364](#).

```
CALL BPX1PST,          Examine and change interrupt type +
      (INTRTYPE,      Input: Interrupt type  BPXYCONS +
      RETVAL,          Return value: 0 or -1
      RETCODE,         Return code
      RSNCODE),        Reason code
      VL,MF=(E,PLIST)  -----
```

## BPX1PTB (pthread\_cancel) example

The following code generates a cancelation request for the target thread (THID). For the callable service, see [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#). AMODE 64 callers use [“BPX4PTB \(pthread\\_cancel\) example” on page 1364](#).

```
CALL BPX1PTB,          pthread_cancel
      (THID,           Input: Thread ID
      RETVAL,          Return Value: 0, -1, or Buf length+
      RETCODE,         Return code
      RSNCODE),        Reason code
      VL,MF=(E,PLIST)  -----
```

## BPX1PTC (pthread\_create) example

The following code creates a new thread. For the callable service, see [“pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread” on page 579](#). For the data structure, see [“BPXYPTAT — Map attributes](#)

for `pthread_exit_and_get` on page 1127. AMODE 64 callers use [“BPX4PTC \(pthread\\_create\) example”](#) on page 1364.

```

LA      R15,BUFFERA      Work area
ST      R15,BUFA         ->above
LA      R15,PTAT         Area mapped by BPXYPTAT
ST      R15,PTATA        ->above
MVC     PTATEYE,=C'BPXYPTAT' Set the eye-catcher
MVC     PTATLENGTH,=A(PATATUSEROFFVAL) Length of structure
MVC     PTATSYSOFFSET,=A(PATATSYSOFFVAL) Sys attr offset
MVC     PTATSYSLLENGTH,=A(PATATSYSLLENVAL) Sys attr length
MVC     PATATUSEROFFSET,=A(0) User attr offset
MVC     PATATUSERLENGTH,=A(0) User attr length
LOAD    EP=INITRTN       Get address of Init Rtn
ST      R0,INITRTNA
SPACE
CALL    BPX1PTC,          +
      (INITRTNA,          +
      BUFA,               +
      PTATA,              +
      THID,               +
      RETVAL,             +
      RETCODE,            +
      RSNCODE),           +
      VL,MF=(E,PLIST)    -----

```

## BPX1PTD (pthread\_detach) example

The following code detaches a thread (THID) in the calling process. For the callable service, see [“pthread\\_detach \(BPX1PTD, BPX4PTD\) — Detach a thread”](#) on page 586. AMODE 64 callers use [“BPX4PTD \(pthread\\_detach\) example”](#) on page 1365.

```

CALL    BPX1PTD,          +
      (THID,              +
      RETVAL,             +
      RETCODE,            +
      RSNCODE),           +
      VL,MF=(E,PLIST)    -----

```

## BPX1PTI (pthread\_testintr) example

The following code causes a cancellation point. For the callable service, see [“pthread\\_testintr \(BPX1PTI, BPX4PTI\) — Cause a cancellation point to occur”](#) on page 618. AMODE 64 callers use [“BPX4PTI \(pthread\\_testintr\) example”](#) on page 1365.

```

CALL    BPX1PTI,          +
      (RETVAl,            +
      RETCODE,            +
      RSNCODE),           +
      VL,MF=(E,PLIST)    -----

```

## BPX1PTJ (pthread\_join) example

The following code gets the termination status of a specified thread (THID). For the callable service, see [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread”](#) on page 592. AMODE 64 callers use [“BPX4PTJ \(pthread\\_join\) example”](#) on page 1365.

```

CALL    BPX1PTJ,          +
      (THID,              +
      =A(0),              +
      RETVAL,             +
      RETCODE,            +
      RSNCODE),           +
      VL,MF=(E,PLIST)    -----

```

## BPX1PTK (pthread\_kill) example

The following code sends a signal to a specified thread (THID). For the callable service, see “pthread\_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread” on page 595. For the data structure, see “BPXYSIGH — Signal constants” on page 1148. AMODE 64 callers use “BPX4PTK (pthread\_kill) example” on page 1366.

```

*      MVC  SIGNAL,=A(SIGALRM#)      Input: SIGALRM      BPXYSIGH
*      MVC  SIGNALOPTIONS,=XL4'00000000' Input: Signal options
*      CALL  BPX1PTK,                  pthread_kill      +
          (THID,                      Input: Thread ID  +
          SIGNAL,                     Input: Signal or 0   BPXYSIGH +
          SIGNALOPTIONS,              Input: Signal options +
          RETVAL,                    Return value: 0 or -1 +
          RETCODE,                   Return code         +
          RSNCODE),                  Reason code         +
          VL,MF=(E,PLIST)            -----

```

## BPX1PTQ (pthread\_quiesce) example

The following code terminates all other pthreads in the caller's process. For the callable service, see “pthread\_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process” on page 598. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 64 callers use “BPX4PTQ (pthread\_quiesce) example” on page 1366.

```

      CALL  BPX1PTQ,                  pthread_quiesce      +
          (=A(QUIESCE_TERM),          Input: Quiesce type   BPXYCONS +
          =A(0),                     Input: User data - Catch data PPSD+
          RETVAL,                    Return value: 0 or -1   +
          RETCODE,                   Return code            +
          RSNCODE),                  Reason code            +
          VL,MF=(E,PLIST)            -----

```

## BPX1PTR (ptrace) example

The following code enables a process (PROCID) to be debugged with ptrace. For the callable service, see “ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging” on page 620. For the data structure, see “BPXYPTRC — Map parameters for ptrace” on page 1127. AMODE 64 callers use “BPX4PTR (ptrace) example” on page 1366.

```

*      MVC  PROCID, Process ID from fork
*      SPACE
*      CALL  BPX1PTR,                  Debug another process +
          (=A(PT_ATTACH),             Input: Request      BPXYPTRC +
          PROCID,                     Input: Process ID    +
          =A(0),                      Input: Address        +
          =A(0),                      Input: Data           +
          =A(0),                      Input: Buffer          +
          RETVAL,                    Return value: 0, -1, or Request +
          RETCODE,                   Return code            +
          RSNCODE),                  Reason code            +
          VL,MF=(E,PLIST)            -----

```

## BPX1PTS (pthread\_self) example

The following code gets the thread ID of the calling thread. For the callable service, see “pthread\_self (BPX1PTS, BPX4PTS) — Query the thread ID” on page 609. AMODE 64 callers use “BPX4PTS (pthread\_self) example” on page 1367.

```

      CALL  BPX1PTS,                  pthread_self      +
          (THID),                     Output: Thread ID   +
          VL,MF=(E,PLIST)            -----

```



## BPX1PTT (pthread\_tag\_np) example

The following code updates the pthread tag. For the callable service, see “pthread\_tag\_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data ” on page 615. AMODE 64 callers use “BPX4PTT (pthread\_tag\_np) example” on page 1367.

```

LA      R15,=CL30'UPDATING MONTH-END STATISTICS'
ST      R15,PT_NEWA
LA      R15,PT_OLD
ST      R15,PT_OLDA
CALL    BPX1PTT,          pthread_tag_np          +
      (=A(30),            Input: Length of New Tag      +
      PT_NEWA,             Input: Address of New Tag     +
      PT_OLDL,            Input: Length of Old Tag      +
      PT_OLDA,            Input: Address to store Old Tag +
      RETVAL,             Return value: 0 or -1         +
      RETCODE,            Return code:                  +
      RSNCODE),           Reason code:                  +
      VL,MF=(E,PLIST)     -----

```

## BPX1PTX (pthread\_exit\_and\_get) example

The following code terminates a thread and creates a new thread. For the callable service, see “pthread\_exit\_and\_get (BPX1PTX, BPX4PTX) — Exit and get a new thread ” on page 588. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 64 callers use “BPX4PTX (pthread\_exit\_and\_get) example” on page 1367.

```

CALL    BPX1PTX,          pthread_exit_and_get      +
      (STATFLD,           Input: Status field        +
      OPTIONS,            Input: Options field       +
      SIGNALREG,          Input: Signal registration usrdta+
      RETVAL,             Return value: 0 or -1 ->BPXYPTXL +
      RETCODE,            Return code                +
      RSNCODE),           Reason code                +
      VL,MF=(E,PLIST)     -----

```

## BPX1PVR (pivot\_root) example

The following code moves the root mount to the /usr/inv/new\_root/put\_old directory and makes /usr/inv/new\_root the new root mount. For the callable service, see “pivot\_root (BPX1PVR, BPX4PVR) — Change the root mount” on page 556. For the data structure, see “BPXYMNTE — Map response and element structure of w\_getmntent” on page 1104. AMODE 64 callers use “BPX4PVR (pivot\_root) example” on page 1368.

```

MVC     NEWROOT(18),=CL18'/usr/inv/new_root'
MVC     NEWROOTLEN,=F'18'
MVC     PUTOLD(26),=CL26'/usr/inv/new_root/put_old'
MVC     PUTOLDLEN,=F'26'
SPACE   ,
CALL    BPX1PVR,          Pivot Root                +
      (NEWROOTLEN,        Input: New Root Path length  +
      NEWROOT,            Input: New Root Pathname     +
      PUTOLDLEN,          Input: Put Old Path length   +
      PUTOLD,             Input: Put Old Pathname      +
      RETVAL,             Return value: 0 or -1         +
      RETCODE,            Return code                  +
      RSNCODE),           Reason code                  +
      VL,MF=(E,PLIST)     -----

```

## BPX1PWD (\_\_passwd, \_\_passwd\_\_applid) example

The following code queries/changes the password of a given user ID. For the callable service, see “\_\_passwd, \_\_passwd\_\_applid (BPX1PWD, BPX4PWD) — Verify or change security information” on page 530. AMODE 64 callers use “BPX4PWD (\_\_passwd, \_\_passwd\_\_applid) example” on page 1368.

```

MVC  USERLEN,=F'8'
MVC  USERNAME(8),=CL8'Myuserid'
MVC  OLDPASLEN,=F'8'
MVC  OLDPASS(8),=CL8'MyOldPwd'
MVC  NEWPASLEN,=F'8'
MVC  NEWPASS(8),=CL8'MyNewPwd'
SPACE
CALL  BPX1PWD,          Query/change user ID password      +
      (USERLEN,         Input: Length of user ID          +
      USERNAME,         Input: User ID                    +
      OLDPASLEN,        Input: Length of old password      +
      OLDPASS,          Input: Old password                +
      NEWPASLEN,        Input: Length of new password      +
      NEWPASS,          Input: New password                +
      RETVAL,           Return value 0 or -1               +
      RETCODE,          Return code                        +
      RSNCODE),         Reason code                        +
      VL,MF=(E,PLIST)  -----

```

## BPX1QCT (msgctl) example

The following code removes the message queue from the system. For the callable service, see “msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations” on page 455. For the data structure, see “BPXYMSG — Map interprocess communication message queues” on page 1107. AMODE 64 callers use “BPX4QCT (msgctl) example” on page 1368.

```

CALL  BPX1QCT,          Message queue control (msgctl)    +
      (MSG_ID,          Input: MessageQueueID             +
      =A(IPC_RMID),     Input: Action to take              BPXYIPC +
      =A(0),            Input: ->MSQID_DS or 0             BPXYMSG +
      RETVAL,           Return value: 0, -1                +
      RETCODE,          Return code                        +
      RSNCODE),         Reason code                        +
      VL,MF=(E,PLIST)  -----

```

## BPX1QDB (querydub) example

The following code obtains the dub status information for the current task. The status indicates whether the current task has already been dubbed, is ready to be dubbed, or cannot be dubbed as a process (or thread). AMODE 64 callers use “BPX4QDB (querydub) example” on page 1369.

```

CALL  BPX1QDB,          Query DUB status for this task    +
      (RETVAL,          Return value: -1 or see BPXYCONS   +
      RETCODE,          Return code                        +
      RSNCODE),         Reason code                        +
      VL,MF=(E,PLIST)  -----

```

## BPX1QGT (msgget) example

The following code creates a private message queue. For the callable service, see “msgget (BPX1QGT, BPX4QGT) — Create or find a message queue” on page 458. For the data structure, see “BPXYMSG — Map interprocess communication message queues” on page 1107. AMODE 64 callers use “BPX4QGT (msgget) example” on page 1369.

```

MVI  S_TYPE,IPC_CREAT+IPC_EXCL      Error if exists
MVI  S_MODE1,0                      Not used
MVI  S_MODE2,S_IRUSR                All read and write permissions

```

```

MVI    S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL   BPX1QGT,          Create a message queue      +
      (=A(IPC_PRIVATE),  Input: Key                  +
      S_MODE,            Input: Creation flags BPXYMODE/IPC+
      RETVAL,            Return value: -1 or msg ID +532200
      RETCODE,           Return code                +
      RSNCODE),          Reason code                +
      VL,MF=(E,PLIST)    -----
SPACE ,
ICM    R15,B'1111',RETVAL Test return value
BNP    PSEUDO            Branch on msgget failure
ST     R15,MSG_ID        Store MSG_ID associated with key

```

## BPX1QRC (msgrcv) example

The following code adds a message to the message queue identified by MSG\_ID. For the callable service, see [“msgrcv \(BPX1QRC, BPX4QRC\) — Receive from a message queue”](#) on page 462. For the data structure, see [“BPXYMSG — Map interprocess communication message queues”](#) on page 1107. AMODE 64 callers use [“BPX4QRC \(msgrcv\) example”](#) on page 1369.

```

LA     R15,BUFFERA      R15 -> Utility buffer
ST     R15,BUFA
USING  MSGBUF,R15
MVC    MSG_TYPE(4),=A(0)
MVC    BUFLINA(4),=A(MSG#LENGTH)
MVC    FLAGS(4),=A(0)    Wait for message
DROP   R15
SPACE ,
CALL   BPX1QSN,          Send a message (msgrcv)      +
      (MSG_ID,           Input: MessageQueueID       +
      BUFA,              Input: ->MSGBUF             +
      PRIMARYALET,       Input: ALET of message buffer +
      BUFLINA,           Input: Length MSGBUF         +
      =A(0),             Input: Message Type         +
      FLAGS,             Input: Flags                +
      RETVAL,            Return value: 0, -1          +
      RETCODE,           Return code                +
      RSNCODE),          Reason code                +
      VL,MF=(E,PLIST)    -----

```

## BPX1QSE (quiesce) example

The following code quiesces file system TESTLIB.FILESYS1, making the files in it unavailable for use. For the callable service, see [“quiesce \(BPX1QSE, BPX4QSE\) — Quiesce a file system”](#) on page 650. AMODE 64 callers use [“BPX4QSE \(quiesce\) example”](#) on page 1370.

```

MVC    FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL   BPX1QSE,          Quiesce a file system        +
      (FSNAME,           Input: File system name (44 char) +
      RETVAL,            Return value: 0, -1, or 4        +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                    +
      VL,MF=(E,PLIST)    -----

```

## BPX1QSN (msgsnd) example

The following code adds a message to the message queue identified by MSG\_ID. For the callable service, see [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue”](#) on page 465. For the data structure, see [“BPXYMSG — Map interprocess communication message queues”](#) on page 1107. AMODE 64 callers use [“BPX4QSN \(msgsnd\) example”](#) on page 1370.

```

LA     R15,BUFFERA      R15 -> Utility buffer
ST     R15,BUFA
USING  MSGBUF,R15
MVC    MSG_TYPE(4),=A(0)

```

## BPX1RCV (recv) example

```
MVC MSG_MTEXT(11),=CL11'QSN MSG TEXT'
MVC BUFLINA(4),=A(15)
MVC FLAGS(4),=A(IPC_NOWAIT)    Don't wait on queue full
DROP R15
SPACE ,
CALL BPX1QSN,                  Send a message (msgsnd)          +
    (MSG_ID,                   Input: MessageQueueID           +
    BUFA,                      Input: ->MSGBUF                BPXYMSG +
    PRIMARYALET,               Input: ALET of message buffer    +
    BUFLINA,                   Input: Length MSGBUF             +
    FLAGS,                     Input: Flags                    BPXYIPC +
    RETVAL,                    Return value: 0, -1               +
    RETCODE,                   Return code                       +
    RSNCODE),                  Reason code                       +
    VL,MF=(E,PLIST)           -----
```

## BPX1RCV (recv) example

The following code issues a `recv` for a socket. `SOCKDESC` was returned previously from a call to either `BPX1SOC` or `BPX1ACP`. For the callable service, see [“recv \(BPX1RCV, BPX4RCV\) — Receive data on a socket and store it in a buffer”](#) on page 680. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYMSGF — Map the message flags”](#) on page 1107. AMODE 64 callers use [“BPX4RCV \(recv\) example”](#) on page 1370.

```
SPACE ,
CALL BPX1RCV,                  Receive data on from a socket  +
    (SOCKDESC,                 Input: Socket Descriptor       +
    =A(L'BUFFERA),             Input: Length of input buffer  +
    BUFFERA,                   Input: Address of input buffer  +
    PRIMARYALET,               Input: Alet of input buffer    +
    MSG_FLAGS,                 Input: Flags                    +
    RETVAL,                    Return value: 0 or -1           +
    RETCODE,                   Return code                       +
    RSNCODE),                  Reason code                       +
    VL,MF=(E,PLIST)           -----
```

## BPX1RDD (readdir) example

The following code reads multiple name entries from the specified directory (`DIRECTDES`). For the callable service, see [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory”](#) on page 656. For the data structure, see [“BPXYDIRE — Map directory entries for readdir”](#) on page 1074. AMODE 64 callers use [“BPX4RDD \(readdir\) example”](#) on page 1371.

```
MVC DIRECTDES,...             Directory descriptor from opendir 55
LA R15,BUFFERA
ST R15,BUFA
MVC BUFLINA,=F'1023'
CALL BPX1RDD,                  Read entries from a directory  +
    (DIRECTDES,                Input: Directory file descriptor +
    BUFA,                      Output: ->buffer                BPXYDIRE +
    PRIMARYALET,               Input: buffer ALET                +
    BUFLINA,                   Input: buffer size                +
    RETVAL,                    Return value: 0, -1, entries read +
    RETCODE,                   Return code                       +
    RSNCODE),                  Reason code                       +
    VL,MF=(E,PLIST)           -----
```

## BPX1RDL (readlink) example

The following code reads the contents of symbolic link `/personnel/templink` into the buffer provided. This will be the path name that was specified when the symbolic link was defined. For the callable service, see [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link”](#) on page 666. AMODE 64 callers use [“BPX4RDL \(readlink\) example”](#) on page 1371.

```
MVC BUFFERB(19),=CL19'/personnel/templink'
MVC BUFLNB,=F'19'
```

```

LA      R15,BUFFERA
ST      R15,BUFA
MVC     BUFLLENB,=F'1023'
SPACE   ,
CALL    BPX1RDL,          Read the value of a symbolic link +
      (BUFLLENB,          Input: Linkname length           +
      BUFFERB,            Input: Link name                 +
      BUFLLENB,          Input: Buffer size - 1023          +
      BUFA,              ->Buffer for symbolic link         +
      RETVAL,            Return value: 0, -1 or char count +
      RETCODE,           Return code                       +
      RSNCODE),          Reason code                       +
      VL,MF=(E,PLIST)    -----

```

## BPX1RDV (readv) example

The following code issues a readv for a socket. SOCKDESC was returned previously from a call to either BPX1SOC or BPX1ACP. For the callable service, see [“readv \(BPX1RDV, BPX4RDV\) — Read data and store it in a set of buffers”](#) on page 673. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYIOV — Map the I/O vector structure”](#) on page 1098. AMODE 64 callers use [“BPX4RDV \(readv\) example”](#) on page 1371.

```

SPACE   ,
LA      R2,BUFFERA
ST      R2,IOV_BASE
LA      R2,L'BUFFERA
ST      R2,IOV_LEN
CALL    BPX1RDV,          Read into a vector of buffers +
      (SOCKDESC,          Input: Socket Descriptor       +
      =A(1),              Input: Number of elements in iov +
      IOV,                Input: Iov containing info      +
      PRIMARYALET,        Input: Alet where iov resides   +
      PRIMARYALET,        Input: Alet of buffers for data +
      RETVAL,            Return value: 0 or -1           +
      RETCODE,           Return code                     +
      RSNCODE),          Reason code                     +
      VL,MF=(E,PLIST)    -----

```

## BPX1RDX (read extlink) example

The following code reads the contents of external symbolic link /personnel/tmpxlink into the buffer provided. This will be the path name that was specified when the external symbolic link was defined. For the callable service, see [“read\\_extlink \(BPX1RDX, BPX4RDX\) — Read an external symbolic link”](#) on page 663. AMODE 64 callers use [“BPX4RDX \(read extlink\) example”](#) on page 1372.

```

MVC     BUFFERB(19),=CL19'/personnel/tmpxlink'
MVC     BUFLLENB,=F'19'
LA      R15,BUFFERA
ST      R15,BUFA
MVC     BUFLLENB,=F'1023'
SPACE   ,
CALL    BPX1RDX,          Read value of an external link +
      (BUFLLENB,          Input: Linkname length           +
      BUFFERB,            Input: Link name                 +
      BUFLLENB,          Input: Buffer size - 1023          +
      BUFA,              ->Buffer for symbolic link         +
      RETVAL,            Return value: 0, -1 or char count +
      RETCODE,           Return code                       +
      RSNCODE),          Reason code                       +
      VL,MF=(E,PLIST)    -----

```

## BPX1RD2 (readdir2) example

The following code reads multiple name entries from the specified directory (DIRECTDES). FUIOCURSOR, set to zero by the BPXYFUIO macro, indicates that the system is to begin reading with the first entry in the directory. For the callable service, see [“readdir2 \(BPX1RD2, BPX4RD2\) — Read an entry from a directory”](#)

## BPX1RED (read) example

on page 659. For the data structure, see “BPXYDIRE — Map directory entries for readdir” on page 1074. AMODE 64 callers use “BPX4RD2 (readdir2) example” on page 1372.

```
MVC  DIRECTDES,...      Directory descriptor from opendir 56
MVC  FUIOID,=CL4'FUIO'  Eye Catcher
MVC  FUIOLEN,=AL4(FUIO#LENGTH) length
LA   R15,BUFFERA        Set address of buffer
ST   R15,FUIOBUFFERADDR for directory data in FUIO
MVC  FUIOIBYTESRW,=F'1023' Max number of bytes to read
SPACE ,
CALL BPX1RD2,           Read directory entries          +
    (DIRECTDES,         Input: Directory file descriptor +
    FUIO,               Input/output: BPXYFUIO          +
    RETVAL,             Return value: 0, -1 or char count+
    RETCODE,            Return code                    +
    RSNCODE),           Reason code                    +
    VL,MF=(E,PLIST)     -----
```

## BPX1RED (read) example

The following code reads 80 bytes from the specified file (FILEDESC) and places them in the area provided (BUFFERA). For the callable service, see “read (BPX1RED, BPX4RED) — Read from a file or socket” on page 652. AMODE 64 callers use “BPX4RED (read) example” on page 1372.

```
MVC  FILEDESC,...      File descriptor          57
LA   R15,BUFFERA        Buffer
ST   R15,BUFA           Buffer address
MVC  BUFLINA,=F'80'     Read buffer length
SPACE ,
CALL BPX1RED,           Read from a file          +
    (FILEDESC,         Input: File descriptor      +
    BUFA,              ->Buffer to read into      +
    PRIMARYALET,       Input: Buffer ALET          +
    BUFLINA,           Input: Number of bytes to read +
    RETVAL,            Return value: 0, -1, or char count+
    RETCODE,           Return code                +
    RSNCODE),          Reason code                +
    VL,MF=(E,PLIST)     -----
```

## BPX1REN (rename) example

The following code changes the directory name of a file from usr/sam to usr/samantha. For the callable service, see “rename (BPX1REN, BPX4REN) — Rename a file or directory” on page 690. AMODE 64 callers use “BPX4REN (rename) example” on page 1373.

```
MVC  BUFFERB(07),=CL07'usr/sam'
MVC  BUFLNB,=F'07'
MVC  BUFFERA(12),=CL12'usr/samantha'
MVC  BUFLINA,=F'12'
SPACE ,
CALL BPX1REN,           Rename a file          +
    (BUFLNB,           Input: Old name length    +
    BUFFERB,           Input: Old name          +
    BUFLINA,           Input: New name length    +
    BUFFERA,           Input: New name          +
    RETVAL,            Return value: 0 or -1     +
    RETCODE,           Return code              +
    RSNCODE),          Reason code              +
    VL,MF=(E,PLIST)     -----
```

## BPX1RFM (recvfrom) example

The following code issues a recv from a socket. SOCKDESC was returned from a previous call, either BPX1SOC or BPX1ACP. For the callable service, see “recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer” on page 683. For the data structures, see “BPXYSOCK — Map

SOCKADDR structure and constants” on page 1151 and “BPXYMSGF — Map the message flags” on page 1107. AMODE 64 callers use “BPX4RFM (recvfrom) example” on page 1373.

```

SPACE ,
MVC   MSG_FLAGS4,MSG_PEEK
CALL  BPX1RFM,          Read from a socket          +
      (SOCKDESC,        Input: Socket Descriptor    +
      =A(L'BUFFERA),    Input: Length of the input buffer +
      BUFFERA,          Input: Address of the input buffer+
      PRIMARYALET,      Input: Alet of the input buffer +
      MSG_FLAGS,        Input: Flags                  +
      =A(L'SOCKADDR),   Input: Length of the socket addr +
      SOCKADDR,         Input: The socket address      +
      RETVAL,           Return value: 0 or -1          +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      VL,MF=(E,PLIST)   -----

```

## BPX1RLA (readlinkat) example

The following code creates the file for the file identified by pathname within the directory represented by the file descriptor FILEFD. Prior to this code excerpt, FILEFD would have been to the file descriptor of the desired directory or to AT\_FDCWD to indicate the current directory. For the callable service, see “readlinkat (BPX1RLA, BPX4RLA) — Read the value of a symbolic link relative to a directory file descriptor” on page 669. AMODE 64 callers use “BPX4RLA (readlinkat) example” on page 1373.

```

MVC   BUFFERB(19),=CL19'/personnel/templink'
MVC   BUFLNB,=F'19'
LA     R15,BUFFERA
ST     R15,BUFA
MVC   BUFLNA,=F'1023'
SPACE ,
CALL  BPX1RLA,          Read the value of a symbolic link +
      (BUFLNB,          Input: Linkname length          +
      BUFFERB,          Input: Link name                +
      BUFLNA,           Input: Buffer size - 1023         +
      BUFA,             ->Buffer for symbolic link       +
      FILEDESC,         Input: Directory file descriptor +
      RETVAL,           Return value: 0, -1 or char count +
      RETCODE,          Return code                     +
      RSNCODE),         Reason code                     +
      VL,MF=(E,PLIST)   -----

```

## BPX1RMD (rmdir) example

The following code removes directory applib/user02. For the callable service, see “rmdir (BPX1RMD, BPX4RMD) — Remove a directory ” on page 703. AMODE 64 callers use “BPX4RMD (rmdir) example” on page 1374.

```

MVC   BUFFERA(13),=CL13'applib/user02'
MVC   BUFLNA,=F'13'
SPACE ,
CALL  BPX1RMD,          Remove a directory              +
      (BUFLNA,          Input: Directory name length    +
      BUFFERA,          Input: Directory to be removed  +
      RETVAL,           Return value: 0 or -1           +
      RETCODE,          Return code                     +
      RSNCODE),         Reason code                     +
      VL,MF=(E,PLIST)   -----

```

## BPX1RMG (resource) example

The following code retrieves system-wide resource measurement data. For the callable service, see “resource (BPX1RMG, BPX4RMG) — Measure resources ” on page 699. For the data structure, see

## BPX2RMS (recvmsg) example

“BPXYRMON — Map resource monitor data” on page 1142. AMODE 64 callers use “BPX4RMG (resource) example” on page 1374.

```
CALL BPX1RMG,          Resource measurement gatherer  +
    (RMONL,           Input: Length of BPXYRMON      +
     RMON,            Input: Buffer, BPXYRMON         +
     RETVAL,          Return value: 0 or -1           +
     RETCODE,         Return code                    +
     RSNCODE),        Reason code                   +
    VL,MF=(E,PLIST)   -----
```

## BPX2RMS (recvmsg) example

The following code issues a recvmsg for a socket. SOCKDESC was returned from a previous call to either BPX1SOC or BPX1ACP. For the callable service, see “recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers ” on page 687. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151, “BPXYMSGF — Map the message flags” on page 1107, “BPXYMSGH — Map the message header” on page 1109, and “BPXYIOV — Map the I/O vector structure” on page 1098. AMODE 64 callers use “BPX4RMS (recvmsg) example” on page 1374.

```
SPACE ,
XC     MSGH(MSGH#LENGTH),MSGH  Clear msgh
LA     R2,SOCKADDR
ST     R2,MSGHNAMEPTR          Store the address of sockaddr
LA     R2,SOCK#LEN+SOCK_SUN#LEN
ST     R2,MSGHNAMELEN
LA     R2,IOV
ST     R2,MSGHIOVPTR
MVI    MSGHIOVNUM,1
LA     R2,BUFFERA
ST     R2,IOV_BASE
LA     R2,L'BUFFERA
ST     R2,IOV_LEN
*
CALL   BPX2RMS,                Receive a message from a socket  +
    (SOCKDESC,                 Input: Socket Descriptor        +
     MSGH,                     Input: Address of BPXYMSGH       +
     MSG_FLAGS,                Input: Flags                     +
     PRIMARYALET,              Input: Alet of the iov           +
     PRIMARYALET,              Input: Alet of the buffers in iov +
     RETVAL,                   Return value: 0 or -1            +
     RETCODE,                  Return code                      +
     RSNCODE),                 Reason code                     +
    VL,MF=(E,PLIST)           -----
```

## BPX1RNA (renameat) example

The following code renames a file lnk03oldfile01.txt relative to directory file descriptors. For the data structure, see “BPXYFCTL — Command values and flags for fcntl” on page 1076. AMODE 64 callers use the “BPX4RNA (renameat) example” on page 1375

```
CALL BPX1RNA,                  Rename a file                x
    (=F'18',                   Length of existing file name  x
     =C'lnk03oldfile01.txt',    Name of exiting file        x
     =F'19',                   Length of new file name       x
     =C'lnk03_newfile02.txt',   Name of new File name       x
     Olddirfd,                 Directory for old file        x
     newdirfd,                 Directory for new file        x
     =A(RENAME_NOREPLACE),      Flags                       x
     retVal,                   x
     my_retCode,                x
     rsnCode),                  x
    VL,MF=(E,PLIST)
```

```
PLIST DS 10A
      BPXYFCTL
```



## BPX1RPH (realpath) example

The following code gets the absolute pathname without dot (.), dot-dot (..), or symbolic links for the input pathname. For the callable service, see “realpath (BPX1RPH, BPX4RPH) — Resolve a path name” on page 677. AMODE 64 callers use “BPX4RPH (realpath) example” on page 1375.

```

MVC  BUFFERA(8),=CL2'..'
MVC  BUFLINA,=F'2'
MVC  BUFLNB,=F'1024'      Resolved pathname return area
SPACE ,
CALL  BPX1RPH,            Resolve pathname          +
      (BUFLINA,           Input: Pathname length    +
       BUFFERA,           Input: Pathname           +
       BUFLNB,           Input: Length resolved name area +
       BUFFERB,           Output: Resolved name buffer +
       RETVAL,           Return value: -1 or length  +
       RETCODE,          Return code                 +
       RSNCODE),         Reason code                 +
      VL,MF=(E,PLIST)    -----

```

## BPX1RW (Pwrite) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). It will start writing at specified offset, 30 bytes from start of the file. To positional read from a file, change the FUIORWIND to indicate FUIO#RD. For the callable service, see “pread() and pwrite() (BPX1RW, BPX4RW) — Read from or write to a file without changing the file pointer” on page 572. AMODE 64 callers use “BPX4RW (Pwrite) example” on page 1375.

```

MVC  FILEDESC,           File descriptor from open
XC   FUIO,FUIO           Zero out Fuio fields
MVC  FUIOID,=CL4'FUIO'   Eye Catcher
MVC  FUIOLEN,=AL4(FUIO#LENGTH) length
LA   R15,BUFFERA         Set address of buffer
ST   R15,FUIOBUFFERADDR  for buffer data in FUIO
MVI  FUIORWIND,FUIO#WRT  Flag to indicate to PWrite
MVC  FUIOIBYTESRW,=F'80' Number of bytes to Write
MVC  FUIOCUR2,=F'30'     Offset to start writing
LA   R15,FUIO            Set address of Fuio
ST   R15,LFUIOPTR        For access to Fuio fields
SPACE ,
CALL  BPX1RW,            PWrite to a file          +
      (FILEDESC,         Input: File descriptor    +
       LFUIOPTR,         Input: Address of FUIO struct +
       PRIMARYALET,      Input: Fuio ALET          +
       FUIOLEN,          Input: Fuio Length        +
       RETVAL,           Return value: -1 or bytes written +
       RETCODE,          Return code                 +
       RSNCODE),         Reason code                 +
      VL,MF=(E,PLIST)    -----

```

## BPX1RWD (rewinddir) example

The following code resets the open directory to the beginning. For the callable service, see “rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning” on page 701. AMODE 64 callers use “BPX4RWD (rewinddir) example” on page 1376.

```

MVC  DIRECTDES,...       File descriptor from opendir      59
CALL  BPX1RWD,           Reposition directory at beginning +
      (DIRECTDES,        Input: Directory file descriptor +
       RETVAL,           Return value: 0 or -1            +
       RETCODE,          Return code                     +
       RSNCODE),         Reason code                     +
      VL,MF=(E,PLIST)    -----

```

## BPX1SA2 (\_\_sigactionset) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see “[\\_\\_sigactionset \(BPX1SA2, BPX4SA2\) — Examine or change a set of signal actions](#)” on page 839. For the data structure, see “[BPXYSIGH — Signal constants](#)” on page 1148. AMODE 64 callers use “[BPX4SA2 \(\\_\\_sigactionset\) example](#)” on page 1376.

```

XR      R15,R15
ST      R15,SSETOPTION_FLAGS
OI      SSETOPTION_FLAGS1,SSET_IGINVALID
LA      R14,1
ST      R14,BUFCNTB
LA      R14,BUFFERA
USING   SSET,R14
MVC
SSETCONSOLMASK,=XL8'0002000000000000'
MVC     SSETFLAGS,=XL4'00000000'
MVC     SSETSAMASK,=XL8'0FFF0F0000000000'
MVC     SSETSAHANDLER,EPADDR
MVC     SSETUSERDATA,=CL4'DATA'
DROP    R14
SPACE   ,
CALL    BPX1SA2,          Examine/change multiple sig acts  +
      (=A(1),             Input: One SSET set              +
      BUFFERA,             Input: Signal set input BPXYSSET  +
      BUFCNTB,             In/Out: Number of array elements  +
      BUFFERB,             Output: Address of output struct  +
      SSETOPTION_FLAGS,    Input: Mapped by BPXYSSET         +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)     -----

```

## BPX1SCT (semctl) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM\_ID semaphore set. For the callable service, see “[semctl \(BPX1SCT, BPX4SCT\) — Perform semaphore control operations](#)” on page 713. For the data structure, see “[BPXYSEM — Map interprocess communication semaphores](#)” on page 1146. AMODE 64 callers use “[BPX4SCT \(semctl\) example](#)” on page 1376.

```

LA      R15,BUFFERA
ST      R15,BUFA
MVC     SEM_NUMBER(4),4      Semaphore number 4 in set
SPACE   ,
CALL    BPX1SCT,             Semaphore control operations  +
      (SEM_ID,               Input: Semaphore set ID        +
      SEM_NUMBER,            Input: Semaphore number (0 based) +
      =A(SEM_GETPID),        Input: Action to take          BPXYSEM +
      BUFA,                  Input: Value | Buffer | Array | 0  +
      RETVAL,                Return value: 0, -1 or value     +
      RETCODE,               Return code                     +
      RSNCODE),              Reason code                     +
      VL,MF=(E,PLIST)        -----

```

## BPX1SDD (setdubdefault) example

The following code sets the dub default setting for the subtasks of the caller to process. For the callable service, see “[set\\_dub\\_default \(BPX1SDD, BPX4SDD\) — Set the dub default service](#)” on page 752. AMODE 64 callers use “[BPX4SDD \(setdubdefault\) example](#)” on page 1377.

```

CALL    BPX1SDD,             Set effective group ID        +
      (=A(DUBPROCESS),       Input: Set Dub Constant BPXYCONS +
      RETVAL,                Return value: 0, 1 or -1         +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      VL,MF=(E,PLIST)        -----

```

## BPX1SEC (\_\_login, \_\_login\_\_applid, \_\_certificate) example

The following code will invoke RACF (or other security product) to create a security environment (ACEE) for the calling process with the identity of JOEUSER. For the callable service, see “[\\_\\_login, \\_\\_login\\_\\_applid, \\_\\_certificate \(BPX1SEC, BPX4SEC\) — Provides an interface to the security product](#)” on page 364. AMODE 64 callers use “[BPX4SEC \(\\_\\_login, \\_\\_login\\_\\_applid, \\_\\_certificate\) example](#)” on page 1377.

```

MVC  USERNLEN,=F'7'
MVC  USERNAME(7),=CL7'JOEUSER'
MVC  OLDPASSLEN,=F'8'
MVC  OLDPASS,=CL8'JOESPASS'
MVC  OPTIONS,=F'0'
SPACE ,
CALL  BPX1SEC,          Create security environment      +
      (=A(SEcurity_CREATE#), Input: Function_code      BPXYCONS +
      SECURITY_USERID#,    Input: ID-Type              BPXYCONS +
      USERNLEN,           Input: UserID Length         +
      USERNAME,           Input: UserID               +
      OLDPASSLEN,         Input: Password Length       +
      OLDPASS,            Input: Password             +
      =A(0),              Input: Holder               +
      =A(0),              Input: Holder               +
      OPTIONS,            Input: Options               +
      RETVAL,             Return value: 0 or -1         +
      RETCODE,            Return code                  +
      RSNCODE),           Reason code                  +
      VL,MF=(E,PLIST)    -----

```

## BPX1SEG (setegid) example

The following code sets the effective group ID of the invoker to 1. For the callable service, see “[setegid \(BPX1SEG, BPX4SEG\) — Set the effective group ID](#)” on page 758. AMODE 64 callers use “[BPX4SEG \(setegid\) example](#)” on page 1378.

```

MVC  GROUPID,=XL4'00000001' Value of new effective ID
SPACE ,
CALL  BPX1SEG,          Set effective group ID           +
      (GROUPID,          Input: Group ID                +
      RETVAL,            Return value: 0 or -1          +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                    +
      VL,MF=(E,PLIST)    -----

```

## BPX1SEL (select) example

The following code issues a select for a previously connected socket. SOCKDESC was returned when the socket was created. In this case, the select is for a single socket for read, write and exception. Do not request waiting. There are no ECBs. For the callable service, see “[select/selectex \(BPX1SEL, BPX4SEL\) — Select on file descriptors and message queues](#)” on page 706. For the data structures, see “[BPXYSOCK — Map SOCKADDR structure and constants](#)” on page 1151 and “[BPXYSEL — Map the select options](#)” on page 1145. AMODE 64 callers use “[BPX4SEL \(select\) example](#)” on page 1378.

```

*      SPACE ,
*      MVC  SELLIST(4),=XL4'81000000'                    +
*                                          Turn on the bit representing sd 0 +
*                                          and sd 7
*      LA   R8,8                                           One more than largest descriptor
*      ST   R8,SOCKDESC                                     Set number of sockets to check
*      CALL BPX1SEL,          Select on a set of sockets   +
*      (SOCKDESC,           Input: Number of file descriptors +
*      =A(4),               Input: Length of read list      +
*      SELLIST,             Input: Read list                +
*      =A(4),               Input: Length of write list     +

```

## BPX1SEU (seteuid) example

SELLIST,	Input: Write list	+
=A(4),	Input: Length of exception list	+
SELLIST,	Input: Exception list	+
=A(0),	Input: Address of Timeout value	+
=A(0),	Input: ECB pointer	+
=A(SEL#BITSFORWARD),	Input: Option - bits forward	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1SEU (seteuid) example

The following code sets the effective user ID of the invoker to 1. For the callable service, see “[seteuid \(BPX1SEU, BPX4SEU\) — Set the effective user ID](#)” on page 760. AMODE 64 callers use “[BPX4SEU \(seteuid\) example](#)” on page 1378.

MVC	USERID,=XL4'00000001'	Value of new effective user ID	
SPACE	,		
CALL	BPX1SEU,	Set effective user ID	+
	(USERID,	Input: User ID	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1SF (send\_file) example

The following code create a parameter list to send the contents of the specified file to the designated socket. to 1. For the callable service, see “[send\\_file \(BPX1SF, BPX4SF\) — Send a file on a socket](#)” on page 730. AMODE 64 callers use “[BPX4SF \(send\\_file\) example](#)” on page 1379.

LA	R5,BUFFERA		
ST	R5,BUFA		
USING	SFPL,R5		
XC	SFPL(SFPL#LENGTH),SFPL	Initialize to nulls (required)	
* NULLS=	no header, no trailer, start at offset 0		
*	MVC	SFFileDes,...	Read from file
*	MVC	SFSocketDes,...	Write to Socket
	MVC	SFFileBytesH,=XL4'FFFFFFFF'	To file end
	MVC	SFFileBytesL,=XL4'FFFFFFFF'	To file end
OI	SFflagByte4,SF_Close	Close socket after write	
SPACE	,		
CALL	BPX1SF,	Send_file	+
	(=A(SFPL#LENGTH),	Input: Length of BPXYSFPL	+
	BUFA,	Input: ->SFPL	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1SGE (setgrent) example

The following code resets the group database to the beginning, so that a subsequent BPX1GGE call will restart the group database search from the first entry. For the callable service, see “[setgrent \(BPX1SGE, BPX4SGE\) — Reset the group database](#)” on page 765. AMODE 64 callers use “[BPX4SGE \(setgrent\) example](#)” on page 1379.

CALL	BPX1SGE,	Reset the group database	+
	(RETVAL),	Return value: 0	+
	VL,MF=(E,PLIST)	-----	

## BPX1SGI (setgid) example

The following code sets the real, effective, and save group IDs to 1. For the callable service, see “[setgid \(BPX1SGI, BPX4SGI\) — Set the group ID](#)” on page 762. AMODE 64 callers use “[BPX4SGI \(setgid\) example](#)” on page 1379.

```

MVC  USERID,=XL4'00000001' Value of new group user ID
SPACE ,
CALL  BPX1SGI,          Set group ID                +
      (GROUPID,         Input: Group ID              +
      RETVAL,           Return value: 0 or -1         +
      RETCODE,          Return code                  +
      RSNCODE),         Reason code                  +
      VL,MF=(E,PLIST)   -----

```

## BPX1SGQ (sigqueue) example

The following code queues a signal (SIGUSR1#) to the process specified by PROCID with a signal value of 0. For the callable service, see “[sigqueue \(BPX1SGQ, BPX4SGQ\) — Queue a signal to a process](#)” on page 848. AMODE 64 callers use “[BPX4SGQ \(sigqueue\) example](#)” on page 1379.

```

SPACE ,
CALL  BPX1SGQ,          Queue a signal to a process   +
      (PROCID,          Input: Process ID             +
      =A(SIGUSR1#),     Input: Signal                BPXYSIGH +
      =A(0),            Input: Signal value           +
      =A(0),            Input: Signal options          +
      RETVAL,           Return value: -1 or 0         +
      RETCODE,          Return code                  +
      RSNCODE),         Reason code                  +
      VL,MF=(E,PLIST)   -----

```

## BPX1SGR (setgroups) example

The following code sets the supplementary group id list to the three gids (000000001, 000000002, 000000003) in BUFFERA. For the callable service, see “[setgroups \(BPX1SGR, BPX4SGR\) — Set the supplementary group IDs list](#)” on page 766. AMODE 64 callers use “[BPX4SGR \(setgroups\) example](#)” on page 1380.

```

LA    R15,BUFFERA
ST    R15,BUFA
MVC   BUFFERA(12),=XL12'000000010000000200000003'
SPACE ,
CALL  BPX1SGR,          Set supplementary groups list  +
      (=A(3),           Input: number of sgids in list +
      BUFA,             Input: address of sgids list  +
      RETVAL,           Return value: -1 or 0         +
      RETCODE,          Return code                  +
      RSNCODE),         Reason code                  +
      VL,MF=(E,PLIST)   -----

```

## BPX1SGT (semget) example

The following code creates a private set of 10 semaphores. For the callable service, see “[semget \(BPX1SGT, BPX4SGT\) — Create or find a set of semaphores](#)” on page 718. For the data structure, see “[BPXYSEM — Map interprocess communication semaphores](#)” on page 1146. AMODE 64 callers use “[BPX4SGT \(semget\) example](#)” on page 1380.

```

MVC   KEY(4),=A(IPC_PRIVATE) Local to this family
MVI   S_TYPE,IPC_CREAT+IPC_EXCL Must not already exist
MVI   S_MODE1,0                Not used
MVI   S_MODE2,S_IRUSR          All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH

```

## BPX1SHT (shutdown) example

```
MVC  NUMB_SEMS(4),=A(10)          10 semaphores this set
SPACE ,
CALL  BPX1SGT,                     Create a set of semaphores      +
      (KEY,                        Input: Semaphore key           +
      NUMB_SEMS,                  Input: Number semaphores in set +
      S_MODE,                     Input: Flags      BPXYMODE / BPXYIPC+
      RETVAL,                     Return value: -1 or Semaphore ID +
      RETCODE,                    Return code                     +
      RSNCODE),                   Reason code                     +
      VL,MF=(E,PLIST)             -----
SPACE ,
ICM   R15,B'1111',RETVAL          Test return value
BNP   PSEUDO                      Branch on semget failure
ST    R15,SEM_ID                  Store SEM_ID associated with key
```

## BPX1SHT (shutdown) example

The following code issues a shutdown to stop socket writes to this socket connection. SOCKDESC was returned from a previous call to BPX1SOC. For the callable service, see “shutdown (BPX1SHT, BPX4SHT) — Shut down all or part of a duplex socket connection ” on page 832. AMODE 64 callers use “BPX4SHT (shutdown) example” on page 1380.

```
SPACE ,
CALL  BPX1SHT,                     Shutdown communication      +
      (SOCKDESC,                  Input: Socket Descriptor     +
      SOCK#SHUTDOWNWRITE,         Input: How - shutdown writes  +
      RETVAL,                     Return value: 0 or -1         +
      RETCODE,                    Return code                     +
      RSNCODE),                   Reason code                     +
      VL,MF=(E,PLIST)             -----
```

## BPX1SIA (sigaction) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see “sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action ” on page 834. For the data structure, see “BPXYISIGH — Signal constants” on page 1148. AMODE 64 callers use “BPX4SIA (sigaction) example” on page 1381.

```
XC    NEWMASK,NEWMASK              Don't block additional signals
LA    R15,NCATCHER                 New catcher (NCATCHER=0,1 ->)
ST    R15,NEWHANDL
LA    R15,OCATCHER                 Old catcher (NCATCHER=0,1 ->)
ST    R15,OLDHANDL
SPACE ,
CALL  BPX1SIA,                     Examine or change signal action +
      (=A(SIGALRM#),              Input: Signal constant  BPXYISIGH +
      NEWHANDL,                   Input: 0, ->0, ->1 or ->catcher +
      NEWMASK,                    Input: 64Bit mask of signals  +
      =A(0),                      Input: Action, BPXYISIGH    +
      OLDHANDL,                   0, ->XL4 (return 0, 1 ->catcher) +
      OLDMASK,                    64 bit mask of signals        +
      OLDFLAGS,                   Action, BPXYISIGH            +
      =A(0),                      Data passed to signal routine +
      RETVAL,                     Return value: 0 or -1         +
      RETCODE,                    Return code                     +
      RSNCODE),                   Reason code                     +
      VL,MF=(E,PLIST)             -----
```

## BPX1SIN (server\_init) example

The following code connects a server address space to WLM as a server manager for the WEB subsystem type, WEB1 subsystem name, and IMWHTTP application environment. For the callable service, see “server\_init (BPX1SIN, BPX4SIN) — Server initialization ” on page 742. AMODE 64 callers use “BPX4SIN (server\_init) example” on page 1381.

```
MVC  SUBSYSTYPE,=CL4'WEB '  WEB Subsystem Type
```

```

MVC SUBSYSNAME,=CL8'WEB1      ' WEB1 Subsystem Name
MVC APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
LA   R15,=F'7'              R15 = 7
ST   R15,PARALLELEU         7 Parallel Execution Units
SPACE ,
CALL BPX1SIN,                Server_init                +
    (=A(SRV_SERVERMGR),      Input: Manager Type (Server Mgr) +
    SUBSYSTYPE,              Input: Subsystem Type          +
    SUBSYSNAME,              Input: Subsystem Type          +
    APPLENV,                 Input: Application Environment  +
    PARALLELEU,              Input: Parallel Eu             +
    RETVAL,                  Return value: 0 or -1           +
    RETCODE,                 Return code                     +
    RSNCODE),                Reason code                     +
    VL,MF=(E,PLIST)          -----
L    R15,RETVAL              Load return value
C    R15,=F'-1'              Test for -1 return
BE   PSEUDO                  Branch on error

```

## BPX1SIP (sigpending) example

The following code retrieves the mask used for pending and blocked signals. For the callable service, see “sigpending (BPX1SIP, BPX4SIP) — Examine pending signals” on page 843. AMODE 64 callers use “BPX4SIP (sigpending) example” on page 1381.

```

CALL BPX1SIP,                Determine pending signals      +
    (SIGRET,                  Signal mask return area (XL8)  +
    RETVAL,                   Return value: 0 or -1          +
    RETCODE,                  Return code                     +
    RSNCODE),                Reason code                     +
    VL,MF=(E,PLIST)          -----

```

## BPX1SLK (shmem\_lock) example

The following code initializes a shared memory resident lock. For the callable service, see “shmem\_lock (BPX1SLK, BPX4SLK) — Shared memory lock service” on page 814. AMODE 64 callers use “BPX4SLK (shmem\_lock) example” on page 1382.

```

XR   R15,R15                 R15 = 0
ST   R15,LOCKATTRADDR        No lock attribute Data
SPACE ,
CALL BPX1SLK,                shmem_lock                    +
    (=A(SLK_INIT),           INPUT: Function Code (Init)      +
    =A(SLK_NORMAL),          INPUT: Request Type (Normal)    +
    =A(SLK_SHARED),          INPUT: Lock Type (Shared)        +
    LOCKADDR,                INPUT: ->user lockword (shared mem+
    LOCKATTRADDR,            INPUT: Address of lock attr area +
    LOCKTOKENADDR,          INPUT: Address of Lock Token      +
    RETVAL,                  Return value: >=0 or -1          +
    RETCODE,                 Return code                     +
    RSNCODE),                Reason code                     +
    VL,MF=(E,PLIST)          -----
L    R15,RETVAL              Load return value
C    R15,=F'-1'              Test for -1 return
BE   PSEUDO                  Branch on error

```

## BPX1SLP (sleep) example

The following code suspends running for 8 seconds or until a signal is delivered (whichever comes first). For the callable service, see “sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time” on page 859. AMODE 64 callers use “BPX4SLP (sleep) example” on page 1382.

```

MVC SECONDS,=F'8'           8 seconds
SPACE ,
CALL BPX1SLP,                Temporarily suspend execution  +
    (SECONDS,                Input: Sleep interval in seconds +

```

## BPX1SMF (smf\_record) example

RETVAL),	Return value: 0 or sleep time	+
VL,MF=(E,PLIST)	-----	

## BPX1SMF (smf\_record) example

The following code tests whether SMF recording is active for a specified SMF record type, and if it is, writes an SMF record. For the callable service, see “[smf\\_record \(BPX1SMF, BPX4SMF\) — Write an SMF record](#)” on page 861. AMODE 64 callers use “[BPX4SMF \(smf\\_record\) example](#)” on page 1382.

MVC	SMF_TYPE,=F'108'	Set SMF record type	
MVC	SMF_SUBTYPE,=F'0'	Set SMF record subtype	
MVC	BUFLINA,=F'0'	Set SMF record length	
MVC	BUFA,=F'0'	Zero SMF record address	
CALL	BPX1SMF,	smf_record	+
	(SMF_TYPE,	SMF record type	+
	SMF_SUBTYPE,	SMF record subtype	+
	BUFLINA,	SMF record length	+
	BUFA,	SMF record address set to zero	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test return value	
BNZ	QUIT	Not recording or error, quit	
SPACE	,		
MVI	BUFFERA,C' '		
MVC	BUFFERA+1(255),BUFFERA	Clear SMF record	
MVI	BUFFERA+1,100	Set length in SMF header	
MVI	BUFFERA+5,108	Set SMF type in SMF header	
MVC	BUFFERA+18(16),=CL16'Here is the data'	Set SMF record	
MVC	SMF_TYPE,=F'108'	Set SMF record type	
MVC	SMF_SUBTYPE,=F'0'	Set SMF record subtype	
MVC	BUFLINA,=F'100'	Set SMF record length	
LA	R15,BUFFERA		
ST	R15,BUFA	Set SMF record address	
CALL	BPX1SMF,	smf_record	+
	(SMF_TYPE,	SMF record type	+
	SMF_SUBTYPE,	SMF record subtype	+
	BUFLINA,	SMF record length	+
	BUFA,	SMF record address	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
QUIT	EQU *		

## BPX2SMS (sendmsg) example

The following code sends a message on a socket. SOCKDESC was returned from a previous call to BPX1SOC. For the callable service, see “[sendmsg \(BPX2SMS, BPX4SMS\) — Send messages on a socket](#)” on page 734. For the data structures, see “[BPXYSOCK — Map SOCKADDR structure and constants](#)” on page 1151, “[BPXYIOV — Map the I/O vector structure](#)” on page 1098, and “[BPXYMSGH — Map the message header](#)” on page 1109. AMODE 64 callers use “[BPX4SMS \(sendmsg\) example](#)” on page 1383.

	XC	MSGH(MSGH#LENGTH),MSGH	Clear msgh	
	LA	R2,SOCKADDR		
	ST	R2,MSGHNAMEPTR	Store the address of sockaddr	
	LA	R2,SOCK#LEN+SOCK_SUN#LEN		
	ST	R2,MSGHNAMELEN		
	LA	R2,IOV		
	ST	R2,MSGHIOVPTR		
	MVI	MSGHIOVNUM,1		
*				
	LA	R2,BUFFERA		
	ST	R2,IOV_BASE		
	LA	R2,16		
	ST	R2,IOV_LEN		
	MVC	BUFFERA(16),=CL16'Here is the data'		
*				
	CALL	BPX2SMS,	Send a message on a socket	+
		(SOCKDESC,	Input: Socket Descriptor	+



MSGH,	Input: Address of BPXYMSGH	+
MSG_FLAGS,	Input: Flags	+
PRIMARYALET,	Input: Alet of the iov	+
PRIMARYALET,	Input: Alet of the buffers in iov	+
RETVL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1SND (send) example

The following code issues a send for a socket. SOCKDESC was returned previously from a call to BPX1SOC. For the callable service, see [“send \(BPX1SND, BPX4SND\) — Send data on a socket”](#) on page 726. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYMSGF — Map the message flags”](#) on page 1107. AMODE 64 callers use [“BPX4SND \(send\) example”](#) on page 1383.

MVC	BUFLINA,=F'16'	
MVC	BUFFERA(16),=CL16'Here is the data'	
SPACE	,	
CALL	BPX1SND,	Send data on a socket
	(SOCKDESC,	Input: Socket Descriptor
	=A(L'BUFFERA),	Input: Length of input buffer
	BUFFERA,	Input: input buffer
	PRIMARYALET,	Input: Alet of input buffer
	MSG_FLAGS,	Input: Flags
	RETVL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1SNS (setns) example

The following code disassociates the current process from its IPC namespace and associates it with an existing IPC namespace referred to by the input file descriptor. In this example, the target file descriptor (TARGFD) would have been returned on the open of the IPC namespace link (/proc/pid/ns/ipc). For the callable service, see [“setns \(BPX1SNS, BPX4SNS\) — Reassociate thread with a namespace”](#) on page 771. For the flag definitions, see [“BPXYCLNP— Map clone syscall parameters”](#) on page 1062. For the AMODE 64 example, see [“BPX4SNS \(setns\) example”](#) on page 1384.

CALL	BPX1SNS,	+
	(TARGFD,	+
	=A(CLONE_NEWIPC),	+
	RETVL,	+
	RETCODE,	+
	RSNCODE),	+
	MF=(E,PLIST)	
L	R15,RETVL	Load return value
C	R15,=F'-1'	Test for -1 return
BE	PSEUDO	Branch on error
...		

## BPX1SOC (socket or socketpair) example

The following code creates a pair of stream sockets in the AF\_UNIX domain. For the callable service, see [“socket or socketpair \(BPX1SOC, BPX4SOC\) — Create a socket or a pair of sockets”](#) on page 865. For the data structure, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151. AMODE 64 callers use [“BPX4SOC \(socket or socketpair\) example”](#) on page 1384.

CALL	BPX1SOC,	Create a socket pair	+
	(=A(AF_UNIX),	Input: Domain of AF_UNIX	+
	=A(SOCK#_STREAM),	Input: Type of socket stream	+
	=A(0),	Input: Protocol of 0	+
	=A(2),	Input: Dimension of 2 for pair	+

## BPX1SOP (semop) example

SOCKETS,	Input: Socket vector for return	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1SOP (semop) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM\_ID semaphore set. For the callable service, see [“semop \(BPX1SOP, BPX4SOP\) — Perform semaphore serialization operations”](#) on page 723. For the data structure, see [“BPXYSEM — Map interprocess communication semaphores”](#) on page 1146. AMODE 64 callers use [“BPX4SOP \(semop\) example”](#) on page 1384.

```
LA    R5,BUFFERA          ->Utility buffer
ST    R5,BUFA
USING SEM_BUF_ELE,R5      ->1st SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(0)   Semaphore number 0
MVC   SEM_OP(2),=AL2(-1)   take the resource
MVC   SEM_FLG(2),=AL2(SEM_UNDO) flags (undo,wait)
LA    R5,SEM#BUFLN(R5)    ->next SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(2)   number 2
MVC   SEM_OP(2),=AL2(1)   release the resource
MVC   SEM_FLG(2),=AL2(IPC_NOWAIT) flags (nowait)
LA    R5,SEM#BUFLN(R5)    ->next SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(8)   number 8
MVC   SEM_OP(2),=AL2(0)   test for no resource
MVC   SEM_FLG(2),=AL2(0)   flags (wait)
SPACE ,
MVC   NUMB_SEM_OPS(4),=AL2(3) number of SEM_BUF_ELE in BUFFERA
SPACE ,
CALL  BPX1SOP,             Semaphore control operations      +
      (SEM_ID,             Input: Semaphore set ID          +
      BUFA,                Input: ->SEM_BUF_ELE            BPXYSEM +
      NUMB_SEM_OPS,        Input: Action to take            +
      RETVAL,              Return value: 0, -1 or value      +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      -----
```

## BPX1SPB (queue\_interrupt) example

The following code uses the queue\_interrupt to return the last signal delivered to the signal interface routine (SIR). For the callable service, see [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered”](#) on page 648. AMODE 64 callers use [“BPX4SPB \(queue\\_interrupt\) example”](#) on page 1385.

CALL BPX1SPB,	Queue the signal	+
(RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1SPE (setpwent) example

The following code resets the user database to the beginning, so that a subsequent BPX1GPE call will restart the user database search from the first entry. For the callable service, see [“setpwent \(BPX1SPE, BPX4SPE\) — Reset the user database”](#) on page 782. AMODE 64 callers use [“BPX4SPE \(setpwent\) example”](#) on page 1385.

CALL BPX1SPE,	Reset the user database	+
(RETVAL),	Return value: 0	+
VL,MF=(E,PLIST)	-----	

## BPX1SPG (setpgid) example

The following code places the invoking process in its own process group (zeros indicate that the process group ID is to be set to the process ID). For the callable service, see “[setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control](#)” on page 777. AMODE 64 callers use “[BPX4SPG \(setpgid\) example](#)” on page 1385.

MVC	PROCID,=A(0)	Process ID - current to leader	
MVC	GROUP,=A(0)	Group ID - current to leader	
SPACE	,		
CALL	BPX1SPG,	Set process group ID for Job Ctl	+
	(PROCID,	Input: Process to be placed in grp	+
	GROUP,	Input: Target group	+
	RETVL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1SPM (sigprocmask) example

The following code changes the signal mask to block signals 1 through 16. For the callable service, see “[sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask](#)” on page 845. For the data structure, see “[BPXYSIGH — Signal constants](#)” on page 1148. AMODE 64 callers use “[BPX4SPM \(sigprocmask\) example](#)” on page 1385.

LA	R15,=XL8'FFFF000000000000'	Block signals 1 thru 16	
ST	R15,NEWMASKA	New mask address	
LA	R15,OLDMASK	Old signal mask	
ST	R15,OLDMASKA	Old mask address	
SPACE	,		
CALL	BPX1SPM,	Examine or change signal mask	+
	(=A(SIG_BLOCK#),	Input: How parameter BPXYSIGH	+
	NEWMASKA,	Input: 0, ->CL8	+
	OLDMASKA,	Input: 0   ->returned mask	+
	RETVL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1SPN (spawn) example

The program ictasma at ict/bin gets control as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. The file descriptor count is set to 0, indicating that the child shall inherit all of the parent's file descriptors. The inheritance area passed is set to all zeros, indicating that the child shall inherit the parent's attributes without change. For the callable service, see “[spawn \(BPX1SPN, BPX4SPN\) — Spawn a process](#)” on page 869. AMODE 64 callers use “[BPX4SPN \(spawn\) example](#)” on page 1386.

	MVC	BUFLINA,=F'16'	
	MVC	BUFFERA(16),=C'ict/bin/ictasma'	
	MVC	ARGCNT,=F'3'	
*			First
	LA	R15,=F'4'	Length
	ST	R15,ARGLLST+00	Length parm list
	LA	R15,=CL4'WK18'	Argument
	ST	R15,ARGSLST+00	Argument address parm list
*			Second
	LA	R15,=F'7'	Length
	ST	R15,ARGLLST+04	Length parm list
	LA	R15,=CL7'DEPT37A'	Argument
	ST	R15,ARGSLST+04	Argument address parm list
*			Third
	LA	R15,=F'22'	Length
	ST	R15,ARGLLST+08	Length parm list
	LA	R15,=CL22'RATE(STD,NOEXC,NOSPEC)'	Argument
	ST	R15,ARGSLST+08	Argument address parm list

## BPX1SPR (setpeer) example

```
*
MVC  ENVCNT,=F'0'      Zero environment args passed
MVC  ENVLENS,=F'0'     Addr of env. data length list
MVC  ENVPARMS,=F'0'    Add of env. data

*
MVC  FDCNT,=F'0'      Zero file descriptors passed
MVC  FDLST,=F'0'      File Descriptor list

*
XC   INHE(INHE#LENGTH),INHE  Clear Inheritance structure
SPACE
CALL BPX1SPN,
      (BUFLNA,          Input: Pathname length      +
       BUFFERA,          Input: Pathname            +
       ARGCNT,           Input: Argument count       +
       ARGLLST,          Input: Argument length list  +
       ARGSLST,          Input: Argument address list +
       ENVCNT,           Input: Environment count    +
       ENVLENS,          Input: Environment length list +
       ENVPARMS,         Input: Environment address list +
       FDCNT,            Input: File descriptor count  +
       FDLST,            Input: File descriptor list   +
       =A(INHE#LENGTH),  Input: Length of Inheritance area +
       INHE,             Input: Inheritance area      +
       RETVAL,           Return value: Child PID or -1 +
       RETCODE,          Return code                  +
       RSNCODE),         Reason code                  +
       VL,MF=(E,PLIST)   -----
```

## BPX1SPR (setpeer) example

The following code issues a setpeer to set up the host address. For the callable service, see [“setpeer \(BPX1SPR, BPX4SPR\) — Preset the peer address associated with a socket”](#) on page 775. For the data structure, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151. AMODE 64 callers use [“BPX4SPR \(setpeer\) example”](#) on page 1387.

```
CALL BPX1SPR,          Select on a set of sockets      +
      (SOCKDESC,        Input: Socket Descriptor      +
       SOCK#LEN+SOCK_SUN#LEN, Input: Length of socket address +
       SOCKADDR,        Input: Socket address          +
       SOCK#SO_SET,     Input: Option - set the address  +
       RETVAL,          Return value: 0 or -1          +
       RETCODE,         Return code                    +
       RSNCODE),        Reason code                    +
       VL,MF=(E,PLIST)  -----
```

## BPX1SPW (server\_pwu) example

The following code puts work to the WLM work queue for the IMWHTTP application environment for transaction class A. For the callable service, see [“server\\_pwu \(BPX1SPW, BPX4SPW\) — Server process work unit”](#) on page 746. AMODE 64 callers use [“BPX4SPW \(server\\_pwu\) example”](#) on page 1387.

```
MVC  APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
MVC  TRXCLASS,=CL8'A      ' Transaction Class A
XR   R15,R15              R15 = 0
ST   R15,CLASSIFYLEN      No Classification Data
ST   R15,APPLDATALEN      No Application Data
ST   R15,FDLISTPTR        No File Descriptor List
SPACE
CALL BPX1SPW,             Server_pwu                    +
      (=A(SRV_PUT_NEWWRK), Input: Function Code (Putwork)  +
       TRXCLASS,           Input: Transaction Class        +
       APPLENV,            Input: Application Environment   +
       CLASSIFYLEN,        Input: Classification Area Length +
       CLASSIFYAREAPTR,    Input: Classification Area Address+
       APPLDATALEN,        Input: Application Data Length   +
       APPLDATAPTR,        Input: Application Data Address  +
       FDLISTPTR,          Input: Mapped by BPXYSFDL        +
       RETVAL,             Return value: 0 or -1            +
       RETCODE,            Return code                      +
       RSNCODE),           Reason code                      +
       VL,MF=(E,PLIST)     -----
L    R15,RETVAL            Load return value
```

```

C      R15,=F'-1'      Test for -1 return
BE     PSEUDO           Branch on error

```

## BPX1SPY (setpriority) example

The following code sets the CPU priority based on the input which and who values. The which value used is PRIO\_PROCESS, which indicates that the priority is to be set by process ID. The who value used is 7, to set the priority for process ID 7. For the callable service, see “setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process” on page 779. AMODE 64 callers use “BPX4SPY (setpriority) example” on page 1387.

```

MVC  PROCID,=XL4'00000007' Process ID to set priority for
MVC  PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL  BPX1SPY,              Set priority value          +
      (=A(PRIO_PROCESS),    Input: Set by Process ID    +
      PROCID,               Input: PID to set priority for +
      PRIORITY,             Input: Priority value to set to +
      RETVAL,               Return value: 0 or -1         +
      RETCODE,              Return code                  +
      RSNCODE),             Reason code                  +
      VL,MF=(E,PLIST)       -----
L     R15,RETVAL            Load return value
C     R15,=F'-1'           Test for -1 return
BE    PSEUDO               Branch on error

```

## BPX1SRG (setregid) example

The following code sets the real and/or effective group IDs to 1. For the callable service, see “setregid (BPX1SRG, BPX4SRG) — Set the real and effective GIDs ” on page 784. AMODE 64 callers use “BPX4SRG (setregid) example” on page 1388.

```

MVC  RGID,=XL4'00000001' Value of new real group ID
MVC  RGID,...           Group ID to be set from a getgid 73
MVC  EGID,=XL4'00000001' Value of new effective group ID
MVC  EGID,...           Group ID to be set from a getegid 73
SPACE ,
CALL  BPX1SRG,          Set Group IDs                  +
      (RGID,            Input: Real Group ID to be set   +
      EGID,             Input: Eff. Group ID to be set   +
      RETVAL,           Return value: 0 or -1            +
      RETCODE,          Return code                      +
      RSNCODE),         Reason code                      +
      VL,MF=(E,PLIST)   -----

```

## BPX1SRL (setrlimit) example

The following code sets the resource limits for the calling process based on the input resource value and the resource limits set in the input rlimit structure. The resource value is set to RLIMIT\_CPU. The resource limits are set to RLIM\_INFINITY. For the callable service, see “setrlimit (BPX1SRL, BPX4SRL) — Set resource limits ” on page 789. For the data structure, see “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142. AMODE 64 callers use “BPX4SRL (setrlimit) example” on page 1388.

```

MVC  RESOURCE,=A(RLIMIT_CPU) Value of resource
XC   RLIM_CUR_HW,RLIM_CUR_HW Current limit highword (Zero)
XC   RLIM_MAX_HW,RLIM_MAX_HW Maximum limit highword (Zero)
MVC  RLIM_CUR,=A(RLIM_INFINITY) Current limit
MVC  RLIM_MAX,=A(RLIM_INFINITY) Maximum limit
SPACE ,
CALL  BPX1SRL,          Set resource limits            +
      (RESOURCE,        Input: resource                +
      RLIM,             Structure, mapped by BPXYRLIM  +
      RETVAL,           Return value: 0 or -1          +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      VL,MF=(E,PLIST)   -----

```

## BPX1SRU (setreuid) example

L	R15, RETVAL	Load return value
C	R15, =F'-1'	Test for -1 return
BE	PSEUDO	Branch on error

## BPX1SRU (setreuid) example

The following code sets the real and/or effective user IDs to 1. For the callable service, see [“setreuid \(BPX1SRU, BPX4SRU\) — Set the real and effective UIDs” on page 786](#). AMODE 64 callers use [“BPX4SRU \(setreuid\) example” on page 1388](#).

MVC	RUID, =XL4'00000001'	Value of new real user ID	
MVC	RUID, ..	User ID to be set from a getuid	73
MVC	EUID, =XL4'00000001'	Value of new effective user ID	
MVC	EUID, ..	User ID to be set from a geteuid	73
SPACE	,		
CALL	BPX1SRU,	Set user IDs	+
	(RUID,	Input: Real User ID to be set	+
	EUID,	Input: Eff. User ID to be set	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL, MF=(E, PLIST)	-----	

## BPX1SRX (srx\_np) example

srx\_np callable service sends or receives data on a socket using CSM buffers. The following example receives data into CSM buffers. The MSGXNAMEPTR is set up to point to a buffer to receive the source address of the data. The MSGXIOVX is an IVTBUFL structure, which describes an IOVX array in a CSM buffer. The IOVX array contains IVTBUFL structures, each of which describes a CSM buffer with data that was received. SOCKDESC is a socket descriptor that was returned from a previous call to either BPX1SOC or BPX1ACP. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) and [“BPXYMSGX — Map the message header” on page 1109](#). For the callable service, see [“srx\\_np \(BPX1SRX, BPX4SRX\) — Send or receive CSM buffers on a socket ” on page 886](#). AMODE 64 callers use [“BPX4SRX \(srx\\_np\) example” on page 1389](#).

XC	MSGX(MSGX#LEN), MSGX	Clear msgx storage	
LA	R2, SOCKADDR		
ST	R2, MSGXNAMEPTR	Store the address of sockaddr	
LA	R2, SOCK#LEN+SOCK_SIN#LEN		
ST	R2, MSGXNAMELEN	Length of sockaddr buffer	
SPACE	,		
CALL	BPX1SRX,	Receive data in CSM buffers	+
	(SOCKDESC,	Input: Socket Descriptor	+
	MSGX_RECV,	Input: Direction	+
	L'MSGX,	Input: Msghdrx length	+
	MSGX,	Input: Msghdrx	+
	RETVAL,	Return value: -1 or bytes read	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL, MF=(E, PLIST)	-----	

## BPX1SSI (setsid) example

The following code creates a session and a process group (and is the leader of both). For the callable service, see [“setsid \(BPX1SSI, BPX4SSI\) — Create a session and set the process group ID ” on page 793](#). AMODE 64 callers use [“BPX4SSI \(setsid\) example” on page 1389](#).

CALL	BPX1SSI,	Create session, set process grp ID+	
	(RETVAL,	Return value: -1 or new session ID+	
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL, MF=(E, PLIST)	-----	

## BPX1SSU (sigsuspend) example

The following code replaces the invoker's current mask to block signals 1 through 16 and suspend until a signal is delivered. For the callable service, see [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered”](#) on page 851. AMODE 64 callers use [“BPX4SSU \(sigsuspend\) example”](#) on page 1389.

```

MVC  WAITMASK(8),=XL8'FFFF000000000000'  Blocks 1 thru 16
SPACE ,
CALL  BPX1SSU,          Wait for a signal          +
      (WAITMASK,        Input: Wait mask, XL8      +
      RETVAL,           Return value: -1 or not returned +
      RETCODE,          Return code                +
      RSNCODE),         Reason code                +
      VL,MF=(E,PLIST)   -----

```

## BPX1STA (stat) example

The following code obtains status about file labrec/qual/current . For the callable service, see [“stat \(BPX1STA, BPX4STA\) — Get status information about a file by path name ”](#) on page 893. For the data structure, see [“BPXYSTAT — Map the response structure for stat ”](#) on page 1162. AMODE 64 callers use [“BPX4STA \(stat\) example”](#) on page 1389.

```

MVC  BUFFERA(19),=CL19'labrec/qual/current'
MVC  BUFLINA,=F'19'
SPACE ,
CALL  BPX1STA,          Get file status            +
      (BUFLINA,         Input: Pathname length    +
      BUFFERA,          Input: Pathname           +
      STATL,           Input: Length of buffer needed +
      STAT,            Buffer, BPXYSTAT           +
      RETVAL,          Return value: 0 or -1      +
      RETCODE,         Return code               +
      RSNCODE),        Reason code               +
      VL,MF=(E,PLIST)   -----

```

## BPX1STE (set\_timer\_event) example

The following code sets a timer event, which when it expires will post the ECB represented by THLITIMERECEB. For the callable service, see [“set\\_timer\\_event \(BPX1STE, BPX4STE\) — Set DIE-mode timer event”](#) on page 798. AMODE 64 callers use [“BPX4STE \(set\\_timer\\_event\) example”](#) on page 1390.

```

CALL  BPX1STE,          Set timer event            +
      (=A(2),           Input: Number of seconds  +
      =A(500000000),    Input: Number of nanoseconds +
      RETVAL,          Return value: 0 or -1      +
      RETCODE,         Return code               +
      RSNCODE),        Reason code               +
      VL,MF=(E,PLIST)   -----

```

## BPX1STF (w\_statvfs) example

The following code obtains information about file system TESTLIB.FILESYS1. For the callable service, see [“w\\_statvfs \(BPX1STF, BPX4STF\) — Get the file system status ”](#) on page 1029. For the data structure, see [“BPXYSSTF — Map response structure for file system status”](#) on page 1161. AMODE 64 callers use [“BPX4STW \(sigtimedwait\) example”](#) on page 1392.

```

MVC  FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX1STF,          Get file system status      +
      (FSNAME,          Input: File system name (44 char) +
      SSTFL,           Input: Length of BPXYSSTF      +
      SSTF,            Buffer, BPXYSSTF              +

```



## BPX1STL (set\_thread\_limits) example

RETVAL,	Return value: -1 or length status	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1STL (set\_thread\_limits) example

The following code sets the MAX\_THREAD and MAX\_THREAD\_TASKS limits for pthread\_created threads in the invoker's process. For the callable service, see [“set\\_thread\\_limits \(BPX1STL, BPX4STL\) — Change task or thread limits for pthread\\_created threads”](#) on page 795. AMODE 64 callers use [“BPX4STL \(set\\_thread\\_limits\) example”](#) on page 1390.

CALL	BPX1STL,	Set_thread_limits	+	
	(=A(STL_SET_BOTH),	Input: action	BPXYCONS	+
	=A(50),	Input: new task limit	+	
	=A(100),	Input: new thread limit	+	
	RETVAL,	Return value: 0 or -1	+	
	RETCODE,	Return code	+	
	RSNCODE),	Reason code	+	
	VL,MF=(E,PLIST)	-----		

## BPX1STO (sendto) example

The following code issues a sendto for a socket. SOCKDESC was returned from a previous call to either BPX1SOC or BPX1ACP. For the callable service, see [“sendto \(BPX1STO, BPX4STO\) — Send data on a socket”](#) on page 738. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYMSGF — Map the message flags”](#) on page 1107. AMODE 64 callers use [“BPX4STO \(sendto\) example”](#) on page 1391.

MVC	BUFFERA(16),=CL16'Here is the data'		
LA	R2,BUFFERA		
ST	R2,IOV_BASE		
MVI	IOV_LEN,16		
SPACE	,		
CALL	BPX1STO,	Send data to a socket	+
	(SOCKDESC,	Input: Socket Descriptor	+
	=A(L'BUFFERA),	Input: Length of the input buffer	+
	BUFFERA,	Input: input buffer	+
	PRIMARYALET,	Input: Alet of the input buffer	+
	MSG_FLAGS,	Input: Flags	+
	=A(L'SOCKADDR),	Input: Length of the socket addr	+
	SOCKADDR,	Input: The socket address	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1STR (setitimer) example

The following code returns the time remaining an alarm, or ITIMER\_REAL as set by setitimer. For the callable service, see [“setitimer \(BPX1STR, BPX4STR\) — Set the value of the interval timer”](#) on page 768. For the data structure, see [“BPXYITIM — Map getitimer, setitimer structure”](#) on page 1101. AMODE 64 callers use [“BPX4STR \(setitimer\) example”](#) on page 1391.

LA	R15,2	Initial value 2.5 seconds	
ST	R15,ITIMISECONDS		
L	R15,=A(500000)		
ST	R15,ITIMIMICROSEC		
L	R15,0	No reload value	
ST	R15,ITIMRSECONDS		
ST	R15,ITIMRMICROSEC		
LA	R15,ITIM	Output mapping structure	
ST	R15,ITIMA	->structure	
CALL	BPX1STR,	Get process data	+
	(=A(ITIMER_REAL),	Input: Relative process token	+
	ITIMA,	In : ->Buffer, mapped by BPXYITIM	+



ITIMA,	Out: ->Buffer, mapped by BPXYITIM	+
RETVAL,	Return value: -1, 0	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1STV (statvfs) example

The following code obtains information about the file system containing the file identified by pathname. For the callable service, see “statvfs (BPX1STV, BPX4STV) — Get the file system status” on page 896. For the data structure, see “BPXYSSTF — Map response structure for file system status” on page 1161. AMODE 64 callers use “BPX4STV (statvfs) example” on page 1391.

MVC	BUFFERA(8),=CL8'/usr/inv'	
MVC	BUFLINA,=F'8'	
SPACE	,	
CALL	BPX1STV,	Get file system status
	(BUFLINA,	Input: Pathname length
	BUFFERA,	Input: Pathname
	SSTFL,	Input: Length of BPXYSSTF
	SSTF,	Buffer, BPXYSSTF
	RETVAL,	Return value: -1 or length status
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1STW (sigtimedwait) example

The following code will wait for signals 1-4 to arrive or 3 seconds, whichever occurs first. For the callable service, see “sigtimedwait (BPX1STW, BPX4STW) — Wait for a signal with a specified timeout” on page 854. AMODE 64 callers use “BPX4STW (sigtimedwait) example” on page 1392.

MVC	WAITMASK(8),=XL8'F000000000000000'	Signals 1-4
LA	R15,SIGINFO_T	
ST	R15,SINFA	
MVC	SECONDS,=F'3'	Wait three seconds
XC	NANOSECONDS,NANOSECONDS	Zero nanoseconds
SPACE	,	
CALL	BPX1STW,	Signal timed wait
	(WAITMASK,	Input: mask of signal to wait for
	SINFA,	Input: address of siginfo_t area
	SIGINFO#LENGTH,	Input: length of siginfo_t area
	SECONDS,	Input: seconds to wait for sig
	NANOSECONDS,	Input: nanoseconds to wait for sig
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1SUI (setuid) example

The following code sets the real, effective, and saved user IDs to 1. For the callable service, see “setuid (BPX1SUI, BPX4SUI) — Set user IDs” on page 800. AMODE 64 callers use “BPX4SUI (setuid) example” on page 1392.

MVC	USERID,=XL4'00000001'	Value of new user ID
MVC	USERID,..	User ID to be set from a getuid
SPACE	,	
CALL	BPX1SUI,	Set user ID
	(USERID,	Input: User ID to be set
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1SWT (sigwait) example

The following code waits for an asynchronous signal, SIGALRM bit 14 in the mask. For the callable service, see [“sigwait \(BPX1SWT, BPX4SWT\) — Wait for a signal ” on page 857](#). For the data structure, see [“BPXYSIGH — Signal constants” on page 1148](#). AMODE 64 callers use [“BPX4SWT \(sigwait\) example” on page 1392](#).

```

MVC WAITMASK(8),=XL8'00040000000000000000'
SPACE ,
CALL BPX1SWT,          Wait for asynchronous signal      +
      (WAITMASK,       Input: Signal mask SIGALRM        +
      RETVAL,          Return value: 0 or -1             +
      RETCODE,         Return code                      +
      RSNCODE),        Reason code                      +
      VL,MF=(E,PLIST)  -----

```

## BPX1SYA (symlinkat) example

The following code creates a symbolic link relative to a directory file descriptor. For the callable service, use [“symlinkat \(BPX1SYA, BPX4SYA\) — Create a symbolic link relative to a directory file descriptor ” on page 905](#). AMODE 64 callers use the [“BPX4SYA \(symlinkat\) example” on page 1392](#).

```

MVC FILEDESC,..        From opendir
MVC BUFFERA(12),=CL12'/sys12/acctn'
MVC BUFLINA,=F'12'
MVC BUFFERB(09),=CL09'/sysaccts'
MVC BUFLINB,=F'09'
SPACE ,
CALL BPX1SYA,          Create symbolic link to pathname +
      (BUFLINA,        Input: Old Pathname length      +
      BUFFERA,         Input: Old Pathname             +
      FILEDESC,        Input: Directory File descriptor +
      BUFLINB,         Input: New Pathname length.     +
      BUFFERB,         Input: New Pathname             +
      RETVAL,          Return value: 0 or -1           +
      RETCODE,         Return code                    +
      RSNCODE),        Reason code                    +
      VL,MF=(E,PLIST)  -----

```

## BPX1SYC (sysconf) example

The following code gets the maximum number of children allowed by the configuration variable. For the callable service, see [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options ” on page 913](#). For the data structure, see [“BPXYCONS — Constants used by services” on page 1062](#). AMODE 64 callers use [“BPX4SYC \(sysconf\) example” on page 1393](#).

```

CALL BPX1SYC,          Get configuration variable      +
      (=A(SC_CHILD_MAX), Input: Config variable BPXYCONS +
      RETVAL,          Return value: -1 or variable    +
      RETCODE,         Return code                    +
      RSNCODE),        Reason code                    +
      VL,MF=(E,PLIST)  -----

```

## BPX1SYF (syncfs) example

The following code synchronizes the file system file referred to by the open file descriptor. For the callable service, see [“syncfs \(BPX1SYF, BPX4SYF\) — Synchronizes the file system file referred to by the open file descriptor ” on page 912](#). AMODE 64 callers use [“BPX4SYF \(syncfs\) example” on page 1393](#).

```

CALL BPX1SYF( FileDesc,          Synfs                +
      RETVAL,          Return value: 0 or -1          +
      RETCODE,         Return code                    +
      RSNCODE),        Reason code                    +

```

VL,MF=(E,PLIST) -----

## BPX1SYM (symlink) example

The following code creates a symbolic link /sysaccts for path name /sys12/acctn. For the callable service, see “symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name ” on page 900. AMODE 64 callers use “BPX4SYM (symlink) example” on page 1393.

```

MVC  BUFFERA(12),=CL12' /sys12/acctn'
MVC  BUFLINA,=F'12'
MVC  BUFFERB(09),=CL09' /sysaccts'
MVC  BUFLINB,=F'09'
SPACE ,
CALL  BPX1SYM,          Create symbolic link to pathname +
      (BUFLINA,          Input: Pathname length          +
      BUFFERA,          Input: Pathname                  +
      BUFLINB,          Input: Link name length          +
      BUFFERB,          Input: Link name                  +
      RETVAL,           Return value: 0 or -1              +
      RETCODE,          Return code                      +
      RSNCODE),         Reason code                      +
      VL,MF=(E,PLIST)  -----

```

## BPX1SYN (sync) example

The following code causes all information in memory that updates file systems to be scheduled for writing out to disk. For the callable service, see “sync (BPX1SYN, BPX4SYN) — Schedule file system updates ” on page 910. AMODE 64 callers use “BPX4SYN (sync) example” on page 1394.

```

CALL  BPX1SYN,          Sync                                +
      (RETVAL,          Return value: 0 or -1              +
      RETCODE,          Return code                      +
      RSNCODE),         Reason code                      +
      VL,MF=(E,PLIST)  -----

```

## BPX1TAF (MVSThreadAffinity) example

The following code executes the assembler routine EXITRTN on another thread, identified by thread ID THID, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see “MVSThreadAffinity (BPX1TAF, BPX4TAF) — MVS thread affinity service” on page 492. AMODE 64 callers use “BPX4TAF (MVSThreadAffinity) example” on page 1394.

```

*      MVC  EXITRTNA,=V(EXITRTN)  ->Routine address
MVC  EXITPLA,=A(EXITPARM)  ->Input parameter list
SPACE ,
CALL  BPX1TAF,
      (EXITRTNA,          Input: Routine address          +
      EXITPLA,          Input: Parm list address or 0      +
      THID,             Input: Target pthread to run exit +
      RETVAL,           Return value: -1 or not return    +
      RETCODE,          Return code                      +
      RSNCODE),         Reason code                      +
      VL,MF=(E,PLIST)  -----

```

## BPX1TAK (takesocket) example

The following code takes a socket that was given by the program identified by CID (clientid). SOCKDESC and CID information are passed by the program that did the givesocket (BPX1GIV). SOCKDESC is the giver's descriptor. When takesocket completes successfully, RETVAL will contain the taker's new socket descriptor. For the callable service, see “takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from

## BPX1TDR (tcdrain) example

another program ” on page 916. For the data structure, see “BPXYCID — Map the returning structure for getclientid()” on page 1061. AMODE 64 callers use “BPX4TAK (takesocket) example” on page 1394.

```
CALL BPX1TAK,          take a socket from another program+
    (CID,              Input: Clientid of giver          +
    SOCKDESC,          Input: Giver's socket descriptor +
    RETVAL,            Return value: -1 or new descriptor+
    RETCODE,           Return code                      +
    RSNCODE),          Reason code                      +
    VL,MF=(E,PLIST)    -----
L   R2,RETVAL
ST  R2,SOCKDES2        Store the new socket descriptor
```

## BPX1TDR (tcdrain) example

The following code waits until all output sent to the standard output file has been transmitted. For the callable service, see “tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted ” on page 918. AMODE 64 callers use “BPX4TDR (tcdrain) example” on page 1394.

```
CALL BPX1TDR,          Wait for output transmittal      +
    (=A(STDOUT_FILENO), Input: File descriptor          +
    RETVAL,            Return value: 0 or -1            +
    RETCODE,           Return code                      +
    RSNCODE),          Reason code                      +
    VL,MF=(E,PLIST)    -----
```

## BPX1TFH (tcflush) example

The following code flushes all the data in the standard input file. the callable service, see “tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal ” on page 923. For the data structure, see “BPXYTIOS — Map the termios structure” on page 1169. AMODE 64 callers use “BPX4TFH (tcflush) example” on page 1395.

```
CALL BPX1TFH,          Line control flush              +
    (=A(STDIN_FILENO), Input: File descriptor          +
    =A(TCIFLUSH),      Input: Queue selector BPXYTIOS  +
    RETVAL,            Return value: 0 or -1            +
    RETCODE,           Return code                      +
    RSNCODE),          Reason code                      +
    VL,MF=(E,PLIST)    -----
```

## BPX1TFW (tcflow) example

The following code resumes data flow (TCION transmits a START character) on the standard input file. For the callable service, see “tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal ” on page 921. For the data structure, see “BPXYTIOS — Map the termios structure” on page 1169. AMODE 64 callers use “BPX4TFW (tcflow) example” on page 1395.

```
CALL BPX1TFW,          Suspend or resume data flow     +
    (=A(STDIN_FILENO), Input: File descriptor          +
    =A(TCION),         Input: Action BPXYTIOS          +
    RETVAL,            Return value: 0 or -1            +
    RETCODE,           Return code                      +
    RSNCODE),          Reason code                      +
    VL,MF=(E,PLIST)    -----
```

## BPX1TGA (tcgetattr) example

The following code retrieves control information about the standard input file. For the callable service, see “tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal ” on page 926. For the data structure,

see “BPXYTIOS — Map the termios structure” on page 1169. AMODE 64 callers use “BPX4TGA (tcgetattr) example” on page 1395.

```
CALL BPX1TGA,          Get a terminal control structure +
    (=A(STDIN_FILENO), Input: File descriptor         +
    TIOS,              Termio structure, BPXYTIOS       +
    RETVAL,            Return value: 0 or -1            +
    RETCODE,           Return code                     +
    RSNCODE),          Reason code                     +
    VL,MF=(E,PLIST)    -----
```

## BPX1TGC (tcgetcp) example

The following code retrieves information about code page change notification (CPCN) capability and the BPXYTCCP structure. For the callable service, see “tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names” on page 928. For the data structure, see “BPXYTCCP — Map the terminal control code page structure” on page 1163. AMODE 64 callers use “BPX4TGC (tcgetcp) example” on page 1395.

```
CALL BPX1TGC,          Get code page names             +
    (=A(STDIN_FILENO), Input: File descriptor         +
    =A(TCCP#LENGTH),   Input: Length of BPXYTCCP      +
    TCCP,              Output: Termcp structure BPXYTCCP +
    RETVAL,            Return value: 0 or -1            +
    RETCODE,           Return code                     +
    RSNCODE),          Reason code                     +
    VL,MF=(E,PLIST)    -----
```

## BPX1TGP (tcgetpgrp) example

The following code gets the foreground process group ID associated with the controlling terminal. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see “tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID” on page 931. AMODE 64 callers use “BPX4TGP (tcgetpgrp) example” on page 1396.

```
CALL BPX1TGP,          Get the foreground process grp ID +
    (=A(STDIN_FILENO), Input: File descriptor         +
    RETVAL,            Return value -1, fgnd proc grp ID +
    RETCODE,           Return code                     +
    RSNCODE),          Reason code                     +
    VL,MF=(E,PLIST)    -----
```

## BPX1TGS (tcgetsid) example

The following code retrieves the process group ID of the session for which the terminal specified by file descriptor is the controlling terminal. For the callable service, see “tcgetsid (BPX1TGS, BPX4TGS) — Get a process group ID for the session leader for the controlling terminal” on page 933. AMODE 64 callers use “BPX4TGS (tcgetsid) example” on page 1396.

```
CALL BPX1TGS,          Get session process group ID    +
    (=A(STDIN_FILENO), Input: File descriptor         +
    RETVAL,            Return value: 0 or -1            +
    RETCODE,           Return code                     +
    RSNCODE),          Reason code                     +
    VL,MF=(E,PLIST)    -----
```

## BPX1TIM (times) example

The following code gathers selected times about the invoker's CPU utilization. For the callable service, see “times (BPX1TIM, BPX4TIM) — Get process and child process times” on page 950. For the data structure,

## BPX1TLS (pthread\_security\_np) example

see “BPXYTIMS — Map the response structure for times” on page 1168. AMODE 64 callers use “BPX4TIM (times) example” on page 1396.

```
CALL BPX1TIM,          Process CPU times          +
    (TIMS,             Input: Buffer              BPXYTIMS +
    RETVAL,            Return value: -1 or clock_t  +
    RETCODE,           Return code                +
    RSNCODE),          Reason code                +
    VL,MF=(E,PLIST)    -----
```

## BPX1TLS (pthread\_security\_np) example

The following code creates a thread-level security environment for the calling thread using the identity specified by the caller. For the callable service, see “pthread\_security\_np, pthread\_security\_applid\_np (BPX1TLS, BPX4TLS) — Create or delete thread-level security” on page 601. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 64 callers use “BPX4TLS (pthread\_security\_np) example” on page 1396.

```
MVC IDENT,=CL8'USERID05'
MVC PASSWORD,=CL7'MYPSWRD'
SPACE
CALL BPX1TLS,          pthread_security_np          +
    (=A(TLS_CREATE_THREAD_SEC#), Input: Func_code  BPXYCONS +
    TLS_IDENTITY_USERID#, Input: Identity_type     BPXYCONS +
    =A(8),              Input: Identity length     +
    IDENT,              Input: Identity            +
    =A(7),              Input: Password length     +
    PASSWORD,           Input: Password            +
    RETVAL,             Return value: 0 or -1      +
    RETCODE,            Return code                +
    RSNCODE),           Reason code                +
    VL,MF=(E,PLIST)    -----
```

## BPX1TRU (truncate) example

The following code truncates the file described by /somedir/somefile.c to a length of 512 bytes. For the callable service, see “truncate (BPX1TRU, BPX4TRU) — Change the size of a file ” on page 953. AMODE 64 callers use “BPX4TRU (truncate) example” on page 1397.

```
MVC BUFFERA(20),=CL20'/somedir/somefile.c'
MVC BUFLINA,=F'20'
MVC NEWLEN(8),=FL8'512'
SPACE
CALL BPX1TRU,          Truncate a file              +
    (BUFLINA,          Input: Pathname length      +
    BUFFERA,           Input: Pathname             +
    NEWLEN,            Input: Length to keep       +
    RETVAL,            Return value: 0 or -1      +
    RETCODE,           Return code                +
    RSNCODE),          Reason code                +
    VL,MF=(E,PLIST)    -----
```

## BPX1TSA (tcsetattr) example

The following code turns off the HUPCL (hang up on last close) bit for the standard input file. For the callable service, see “tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal ” on page 937. For the data structure, see “BPXYTIOS — Map the termios structure” on page 1169. AMODE 64 callers use “BPX4TSA (tcsetattr) example” on page 1397.

```
*      NI      C_CFLAG+HUPCL_0,X'FF'-HUPCL  Turn off HUPCL
*      termios was retrieved by a prior tcgetattr
CALL BPX1TSA,          Set terminal attributes      +
    (=A(STDIN_FILENO), Input: File descriptor     +
    =A(TCSADRAIN),      Input: Action              BPXYTIOS +
```

TIOS,	Input: Terminos struct	BPXYTIOS	+
RETVAL,	Return value: 0 or -1		+
RETCODE,	Return code		+
RSNCODE),	Reason code		+
VL,MF=(E,PLIST)	-----		

## BPX1TSB (tcsendbreak) example

The following code requests that a break be sent to the standard input file. For the callable service, see [“tcsendbreak \(BPX1TSB, BPX4TSB\) — Send a break condition to a terminal”](#) on page 934. AMODE 64 callers use [“BPX4TSB \(tcsendbreak\) example”](#) on page 1397.

CALL	BPX1TSB,	Send break condition to terminal	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	=A(0),	Duration, not used in z/OS UNIX	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1TSC (tcsetcp) example

The following code sets code page names and Code Page Change Notification (CPCN) capability. For the callable service, see [“tcsetcp \(BPX1TSC, BPX4TSC\) — Set terminal code page names”](#) on page 940. For the data structure, see [“BPXYTCCP — Map the terminal control code page structure”](#) on page 1163. AMODE 64 callers use [“BPX4TSC \(tcsetcp\) example”](#) on page 1397.

XC	TCCP(TCCP#LENGTH),TCCP	Clear area	
OI	TCCPFLAGB4,TCCPFASTP	Set local translation	
MVC	TCCPSRCNAME(8),=CL8'IBM-1047'	Set source code page name	
MVC	TCCPTRGNAME(9),=CL9'ISO8859-1'	Set target code page name	
SPACE			
CALL	BPX1TSC,	Set code page names	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	=A(TCCP#LENGTH),	Input: Length of BPXYTCCP	+
	TCCP,	Termcp structure, BPXYTCCP	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1TSP (tcsetpgrp) example

The following code sets the controlling terminal's foreground process group to a new value. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see [“tcsetpgrp \(BPX1TSP, BPX4TSP\) — Set the foreground process group ID”](#) on page 944. AMODE 64 callers use [“BPX4TSP \(tcsetpgrp\) example”](#) on page 1398.

MVC	PROCID,..	Process group ID set by setpgrp	
SPACE			
CALL	BPX1TSP,	Set foreground process group ID	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	PROCID,	Input: Foreground process group ID	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX1TST (tcsettables) example

The following code sets code page names, conversion tables and Code Page Change Notification (CPCN) capability. For the callable service, see [“tcsettables \(BPX1TST, BPX4TST\) — Set terminal code page names and conversion tables”](#) on page 946. For the data structure, see [“BPXYTCCP — Map the terminal](#)

## BPX1TYN (ttyname) example

control code page structure” on page 1163. AMODE 64 callers use “BPX4TST (tcsettables) example” on page 1398.

```
XC      TCCP(TCCP#LENGTH),TCCP  Clear area
OI      TCCPFLAGB4,TCCPFASTP  Set local translation
MVC     TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
MVC     TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
MVC     TBLSOURCE,..          Initialize source conversion table
MVC     TBLTARGET,..         Initialize target conversion table
SPACE  ,
CALL    BPX1TST,              Set code page names and tables      +
      (=A(STDIN_FILENO),      Input: File descriptor            +
      =A(TCCP#LENGTH),        Input: Length of BPXYTCCP          +
      TCCP,                    Termcp structure, BPXYTCCP         +
      TBLSOURCE,              Source conversion table             +
      TBLTARGET,              Target conversion table             +
      RETVAL,                  Return value: 0 or -1               +
      RETCODE,                 Return code                         +
      RSNCODE),                Reason code                        +
      VL,MF=(E,PLIST)         -----
```

## BPX1TYN (ttyname) example

The following code retrieves the pathname for the standard error output file. For the callable service, see “ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal ” on page 956.

```
MVC     BUFLINA,=A(1023)      Maximum pathname
CALL    BPX1TYN,              Determine terminal name              +
      (=A(STDERR_FILENO),      Input: File descriptor            +
      BUFLINA,                  Length of buffer for pathname      +
      BUFFERA),                Buffer for pathname of terminal      +
      VL,MF=(E,PLIST)         -----
```

## BPX2TYN (ttyname) example

The following code retrieves the pathname for the standard error output file. For the callable service, see “ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal ” on page 956. AMODE 64 callers use “BPX4TYN (ttyname) example” on page 1399.

```
MVC     BUFLINA,=A(1023)      Maximum pathname
CALL    BPX2TYN,              Determine terminal name              +
      (=A(STDERR_FILENO),      Input: File descriptor            +
      BUFLINA,                  Length of buffer for pathname      +
      BUFFERA),                Buffer for pathname of terminal      +
      RETVAL,                  Return value: 0, -1                 +
      RETCODE,                 Return code: describes why VAL=-1  +
      RSNCODE),                Reason code: qualifier on RETCODE  +
      VL,MF=(E,PLIST)         -----
```

## BPX1ULA (unlinkat) example

The following code removes path name `usr/dataproc/next.t` from the directory that was specified by DIRFD. For the callable service, see “unlinkat (BPX1ULA, BPX4ULA) — Remove a directory or directory entry relative to a directory file descriptor” on page 969. AMODE 64 callers use “BPX4ULA (unlinkat) example” on page 1399.

```
MVC     BUFFERA(19),=CL19'usr/dataproc/next.t'
MVC     BUFLINA,=F'19'
MVC     DIRFD,..              Directory descriptor from opendir
MVC     FLAGS,=F'0'           Flags defined in BPXYFCTL
SPACE  ,
CALL    BPX1ULA,              Remove a directory entry            +
      (BUFLINA,                Input: Pathname length            +
      BUFFERA,                  Input: Pathname                  +
      DIRFD),                  Input: Directory descriptor        +
      VL,MF=(E,PLIST)         -----
```



FLAGS,	Input: Flags	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX1UMK (umask) example

The following code changes the process's file mode creation mask (to user read, group execute, other execute). For the callable service, see [“umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask”](#) on page 960. For the data structure, see [“BPXYMODE — Map the mode constants”](#) on page 1106. AMODE 64 callers use [“BPX4UMK \(umask\) example”](#) on page 1399.

XC	S_MODE,S_MODE	
MVI	S_MODE3,S_IXUSR+S_IXGRP+S_IXOTH	Search permission
SPACE		
CALL	BPX1UMK,	Set file creation mask
	(S_MODE,	Input: Mode
	RETVAL),	Return value: previous mode mask
	VL,MF=(E,PLIST)	-----

## BPX1UMT (umount) example

The following code removes virtual file system TESTLIB.FILESYS1 from the file tree. For the callable service, see [“umount \(BPX1UMT, BPX4UMT\) — Remove a virtual file system”](#) on page 961. For the data structure, see [“BPXYMTM — Map the modes for mount and unmount”](#) on page 1110. AMODE 64 callers use [“BPX4UMT \(umount\) example”](#) on page 1399.

MVC	FSNAME(44),=CL44'TESTLIB.FILESYS1'	
XC	MTM(MTM#LENGTH),MTM	
MVI	MTM1,MTMUMOUNT	Unmount request
SPACE		
CALL	BPX1UMT,	Remove a virtual file system
	(FSNAME,	Input: File system name (44 char)
	MTM,	Input: Flags, BPXYMTM
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1UNA (uname) example

The following code obtains information about the system on which the invoker is running. For the callable service, see [“uname \(BPX1UNA, BPX4UNA\) — Obtain the name of the current operating system”](#) on page 964. For the data structure, see [“BPXYUTSN — Map the response structure for uname”](#) on page 1172. AMODE 64 callers use [“BPX4UNA \(uname\) example”](#) on page 1400.

LA	R15,UTSN	
ST	R15,UTSNA	
SPACE		
CALL	BPX1UNA,	Identify system
	(UTSNL,	Input: Length of required buffer
	UTSNA,	Output: ->UTSN
	RETVAL,	Return value: -1 or >-1
	RETCODE,	Return code
	RSNCODE),	Reason code
	VL,MF=(E,PLIST)	-----

## BPX1UNL (unlink) example

The following code removes path name `usr/dataproc/next.t` from the directory specified by DIRFD. For the callable service, see [“unlink \(BPX1UNL, BPX4UNL\) – Remove a directory entry”](#) on page 966.

```

MVC  BUFFERA(19),=CL19'usr/dataproc/next.t'
MVC  BUFLINA,=F'19'
SPACE ,
CALL  BPX1UNL,          Remove a directory entry      +
      (BUFLINA,          Input: Pathname length        +
      BUFFERA,           Input: Pathname               +
      RETVAL,            Return value: 0 or -1          +
      RETCODE,           Return code                   +
      RSNCODE),          Reason code                   +
      VL,MF=(E,PLIST)   -----

```

## BPX1UPT (unlockpt) example

The following code unlocks the subsidiary pseudoterminal device associated with the manager pseudoterminal to which the file descriptor refers. For the callable service, see [“unlockpt \(BPX1UPT, BPX4UPT\) – Unlock a pseudoterminal manager and subsidiary pair”](#) on page 973. AMODE 64 callers use [“BPX4UPT \(unlockpt\) example”](#) on page 1400.

```

CALL  BPX1UPT,          Unlocks subsidiary pty from manager +
      (MASTER_FD,       Input: File descriptor           +
      RETVAL,           Return value: 0 or -1            +
      RETCODE,          Return code                     +
      RSNCODE),         Reason code                     +
      VL,MF=(E,PLIST)   -----

```

## BPX1UQS (unquiesce) example

The following code unquiesces TESTLIB.FILESYS1, making its files available for use again. For the callable service, see [“unquiesce \(BPX1UQS, BPX4UQS\) – Unquiesce a file system”](#) on page 975. For the data structure, see [“BPXYMTM – Map the modes for mount and unmount”](#) on page 1110. AMODE 64 callers use [“BPX4UQS \(unquiesce\) example”](#) on page 1400.

```

MVC  FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC   MTM(MTM#LENGTH),MTM  Zero MTM = don't force unquiesce
SPACE ,
CALL  BPX1UQS,          Unquiesce a file system          +
      (FSNAME,           Input: File system name (44 char) +
      MTM,               Input: Flags, BPXYMTM            +
      RETVAL,            Return value: 0 or -1            +
      RETCODE,           Return code                     +
      RSNCODE),         Reason code                     +
      VL,MF=(E,PLIST)   -----

```

## BPX1UNS (unshare) example

The following code disassociates the current process from its IPC namespace and associates it with a newly created IPC namespace. For the callable service, see [“unshare \(BPX1UNS, BPX4UNS\) – Disassociate from shared execution resources. For the flag definitions, see “BPXYCLNP— Map clone syscall parameters” on page 1062. For the AMODE 64 example, see “BPX4UNS \(unshare\) example” on page 1401.](#)

```

CALL  BPX1UNS,          +
      (=A(CLONE_NEWIPC), +
      RETVAL,           +
      RETCODE,          +
      RSNCODE),         +

```

```

MF=(E,PLIST)
L      R15,RETVAL      Load return value
C      R15,=F'-1'      Test for -1 return
BE     PSEUDO          Branch on error
...

```

## BPX1UTI (utime) example

The following code changes the access and modification times of /usr/private/workfile.t to the current time. For the callable service, see “[utime \(BPX1UTI, BPX4UTI\) — Set file access and modification times](#)” on page 981. AMODE 64 callers use “[BPX4UTI \(utime\) example](#)” on page 1401.

```

MVC    BUFFERA(23),=CL23'/usr/private/workfile.t'
MVC    BUFLINA,=F'23'
MVC    NEWTIMES,=FL8'-1'      Current time
SPACE  ,
CALL   BPX1UTI,              Set file access and modify times  +
      (BUFLINA,              Input: Pathname length          +
      BUFFERA,               Input: Pathname                  +
      NEWTIMES,              Input: Access/Modification time  +
      RETVAL,                Return value: 0 or -1             +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      VL,MF=(E,PLIST)        -----

```

## BPX1WAT (wait) example

The following code waits for any of its children to end or stop. For the callable service, see “[wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end](#)” on page 984. For the data structure, see “[BPXYWAST — Map the wait status word](#)” on page 1172 and “[BPXYCONS — Constants used by services](#)” on page 1062. AMODE 64 callers use “[BPX4WAT \(wait\) example](#)” on page 1401.

```

LA     R15,WAST             Resolve address of STATUS
ST     R15,WASTA            Save address of STATUS
MVC    PROCID,=F'-1'       Wait for any child
SPACE  ,
CALL   BPX1WAT,             Wait for a child process to end  +
      (PROCID,              Input: PID being waited on      +
      =A(WNOHANG),          Input: options                    +
      WASTA,                ->Exit status field, BPXTWAST      +
      RETVAL,               Return value: -1, 0, child PID    +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)        -----

```

## BPX1WLM (\_\_WLM) example

The following code connects to WLM as a work manager for the WEB subsystem type and WEB1 subsystem name. For the callable service, see “[\\_\\_wlm \(BPX1WLM, BPX4WLM\) — WLM interface service](#)” on page 1018. AMODE 64 callers use “[BPX4WLM \(\\_\\_WLM\) example](#)” on page 1402.

```

LA     R8,BUFFERA           Storage for __WVC
USING  __WVC,R8             WLM_CONNECT_WORKMGR DSECT
ST     R8,INARGLISTPTR      ->__WVC list of parameters
MVC    SUBSYSTYPE,=CL4'WEB ' WEB Subsystem Type
MVC    SUBSYSNAME,=CL8'WEB1 ' WEB1 Subsystem Name
LA     R15,SUBSYSTYPE
ST     R15,__WVC_SUB_SYS    Pointer to Subsystem Type
LA     R15,SUBSYSNAME
ST     R15,__WVC_SUB_SYS_NM Pointer to Subsystem Name
SPACE  ,
CALL   BPX1WLM,             work_load_manager system call  +
      (=A(WLM_CONNECT_WORKMGR), Input: Fcn Codes in BPXYWLM  +
      INARGLISTPTR,         Input: ->list of parameters      +
      RETVAL,                Return value: Varies with fcn code+
      RETCODE,               Return code                      +
      RSNCODE),             Reason code                      +

```

```

          VL,MF=(E,PLIST)
DROP     R8

```

## BPX1WRT (write) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). For the callable service, see “write (BPX1WRT, BPX4WRT) — Write to a file or a socket” on page 1032. AMODE 64 callers use “BPX4WRT (write) example” on page 1402.

```

*      MVC  FILEDESC,          File descriptor from open
      MVC  BUFLINA,=F'80'
      LA   R15,BUFFERA
      ST   R15,BUFA
      SPACE
      CALL BPX1WRT,           Write to a file
      (FILEDESC,             Input: File descriptor
      BUFA,                   Input: ->Buffer
      PRIMARYALET,            Input: Buffer ALET
      BUFLINA,                Input: Number of bytes to write
      RETVAL,                 Return value: -1 or bytes written
      RETCODE,                Return code
      RSNCODE),              Reason code
      VL,MF=(E,PLIST)

```

## BPX1WRV (writev) example

The following code issues a writev for a socket. SOCKDESC was returned from a previous call to either BPX1SOC or BPX1ACP. For the callable service, see “writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers” on page 1036. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151 and “BPXYIOV — Map the I/O vector structure” on page 1098. AMODE 64 callers use “BPX4WRV (writev) example” on page 1402.

```

      MVC  BUFFERA(16),=CL16'Here is the data'
      LA   R2,BUFFERA
      ST   R2,IOV_BASE
      MVI  IOV_LEN,16
*
      CALL BPX1WRV,           Write from a vector of buffers
      (SOCKDESC,             Input: Socket Descriptor
      =A(1),                  Input: Single element in iov
      IOV,                    Input: Iov containing info
      PRIMARYALET,            Input: Alet where iov resides
      PRIMARYALET,            Input: Alet of buffers for data
      RETVAL,                 Return value: 0 or -1
      RETCODE,                Return code
      RSNCODE),              Reason code
      VL,MF=(E,PLIST)

```

## BPX1WTE (wait extension) example

The following code uses the #WAIT3 function to wait for any of its children to end or stop. For the callable service, see “wait-extension (BPX1WTE, BPX4WTE) — Obtain status information for children” on page 987. For the data structures, see “BPXYWAST — Map the wait status word” on page 1172 and “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142. and “BPXYCONS — Constants used by services” on page 1062. AMODE 64 callers use “BPX4WTE (wait extension) example” on page 1403.

```

      LA   R15,WAST           Resolve address of WAST
      ST   R15,WASTA          Save address of WAST
      LA   R15,RUSAGE         Resolve address of RUSAGE
      ST   R15,RUSAGEA        Save address of RUSAGE
      SPACE
      CALL BPX1WTE,           Wait for a child process to end
      (=A(#WAIT3),           Input: function BPXYCONS
      0,                      Input: id type
      0,                      Input: id
      WASTA,                  ->Exit status field, BPXTWAST

```

=A(WNOHANG),	Input: options	BPXYCONS	+
RUSAGEA,	->Rusage structure,	BPXYRLIM	+
RETVAL,	Return value: -1, 0, child PID		+
RETCODE,	Return code		+
RSNCODE),	Reason code		+
VL,MF=(E,PLIST)	-----		

## Reentrant return linkage

XR	R15,R15	Zero return code
L	R0,@SIZEDAT	Size this program's getmain area
LR	R1,R13	R1 -> this program's getmain area
L	R13,@BACK	R2 -> caller's save area
DROP	R13	
FREE	MAIN RU,LV=(0),A=(1)	
L	R14,12(,R13)	Restore caller's R14
LM	R0,R12,20(R13)	Restore caller's R0-R12
BSM	0,R14	Branch back to caller

SPACE	,	*****	Program constants	*****
@SIZEDAT	DC	A(@ENDSTOR-@STORE)	Size of this getmain storage	
MNTEL	DC	A(MNTE#LENGTH+MNTEH#LENGTH)		
*			Length of MNTEH and 1 MNTE area	
PGPSL	DC	A(PGPS#LENGTH)	Length of PGPS structure	
RMONL	DC	A(RMON#LENGTH)	Length of RMON structure	
SSTFL	DC	A(SSTF#LENGTH)	Length of SSTF structure	
STATL	DC	A(STAT#LENGTH)	Length of STAT structure	
UTSNL	DC	A(UTSN#LENGTH)	Length of UTSN structure	
SPACE	,			
PRIMARYALET	DC	A(0)	Primary ALET	

*****	*****	*****	Structures requiring a USING	*
BPXYDIRE	DSECT=	YES	Dictionary for readdir	
BPXYGIDN	DSECT=	YES	Group names	
BPXYGIDS	DSECT=	YES	Group IDs and member names	
BPXYOSMF	DSECT=	YES	Job step accounting for BPXESMF	
BPXYPOLL	DSECT=	YES	Poll syscall parameters (I/O)	
BPXYPGTH	DSECT=	YES	Mapping for __getthent data	
BPXYPPSD	DSECT=	YES	Signal data area (R1 in SIR)	
BPXYSEL	DSECT=	YES	Select options	
BPXYSFPL	DSECT=	YES	Send_file parameter list	
BPXYTHDQ	DSECT=	YES	Data structure for BPX1PQG	
BPXYWLM	,		Work load manager	
*****	*****	*****	EQUates	*****
*	With EQUate only macros,	DSECT=	is allowed but is ignored	
BPXYCONS	,		z/OS UNIX constants	
BPXYCW	,		Serialization constants	
BPXYERNO	LIST=	NO	Errno, Errnojr constants	
BPXYFTYP	,		File type constants	
BPXYPCF	,		Command, pathconf constants	
BPXYSEEK	,		lseek constants	
BPXYSIGH	,		Signal constants	

*****	*****	*****	Standard linkage save area	*
@STORE	DSECT	,		
@SAVE00	DS	0D	Standard 72-byte save area	
	DS	A		
@BACK	DS	A	Back to caller's save area	
@FORWARD	DS	A	Forwards to callee's save area	
	DS	15A	Regs 14,15,0-12	

*****	*****	*****	Standard linkage save area	*
@STORE	DSECT	,		
@SAVE00	DS	0D	Standard 72-byte save area	
	DS	A		
@BACK	DS	A	Back to caller's save area	
@FORWARD	DS	A	Forwards to callee's save area	
	DS	15A	Regs 14,15,0-12	

## Reentrant return linkage

* *			Standard linkage save area	* *
@STORE	DSECT	,		
@SAVE00	DS	0D	Standard 72-byte save area	
	DS	A		
@BACK	DS	A	Back to caller's save area	
@FORWARD	DS	A	Forwards to callee's save area	
	DS	15A	Regs 14,15,0-12	

```
* * * * * * * * * * * * * * .** Standard linkage save area * *
@STORE    DSECT ,
@SAVE00   DS      0D          Standard 72-byte save area
           DS      A
@BACK     DS      A          Back to caller's save area
@FORWARD  DS      A          Forwards to callee's save area
           DS      15A       Regs 14,15,0-12
```

```
* * * * * * * * * * * * * * * * Standard linkage save area * *
@STORE    DSECT ,
@SAVE00   DS    0D           Standard 72-byte save area
          DS      A
@BACK     DS      A         Back to caller's save area
@FORWARD  DS      A         Forwards to callee's save area
          DS    15A        Regs 14,15,0-12
```

* *			Standard linkage save area	* *
@STORE	DSECT	,		
@SAVE00	DS	0D	Standard 72-byte save area	
	DS	A		
@BACK	DS	A	Back to caller's save area	
@FORWARD	DS	A	Forwards to callee's save area	
	DS	15A	Regs 14,15,0-12	

## Appendix E. Callable services examples—AMODE 64

For an example using nonreentrant code, see [“Example of nonreentrant entry linkage—AMODE 64” on page 1409](#). These examples follow the rules of reentrancy. They use DSECT=NO and place the variables in the program's dynamic storage DSECT, which is allocated upon entry.

The examples are arranged alphabetically and have references to the mapping macros they use. The declaration for all local variables used in the examples follows the examples.

### Reentrant entry linkage

This entry linkage is reentrant and saves the caller's registers, allocates a save area and dynamic storage, and establishes program and dynamic storage base registers. This entry linkage is paired with the return linkage that is located at the end of the executable program; see [“Reentrant return linkage” on page 1403](#). For an example of nonreentrant entry and return linkage, see [“Example of nonreentrant entry linkage—AMODE 31” on page 1407](#).

```

        TITLE 'Alphabetical syscall of z/OS UNIX callable services'
BPXB1SM1 CSECT ,           Reentrant entry linkage
BPXB1SM1 AMODE 64
        SYSSTATE AMODE64=YES
BPXB1SM1 RMODE ANY
@ENTRY0 J    @ENTRY1      Branch around program header
        DC    C'BPXB1SM4 - Reentrant callable service examples'
        DS    0H          Ensure half word boundary
@ENTRY1 STMG  R14,R12,12(R13) Save caller's registers
        LGR   R2,R13      Hold address of caller's area
        LGR   R3,R1       Hold parameter register
        LGR   R12,R15     R12 program base register
        LA    R11,2048(,R12) Second program base register
        LA    R11,2048(,R11) Second program base register
        LA    R9,2048(,R11) Third program base register
        LA    R9,2048(,R9)  Third program base register
        LA    R4,2048(,R9)  Fourth program base register
        LA    R4,2048(,R4)  Fourth program base register
        LA    R7,2048(,R4)  Fifth program base register
        LA    R7,2048(,R7)  Fifth program base register
        USING @ENTRY0,R12,R11,R9,R4,R7 Program addressability
        L     R0,@SIZEDAT Size this program's getmain area
        GETMAIN RU,LV=(0)  Getmain storage
        LGR   R13,R1      R13 -> this program's save area
        LA    R10,2048(,R13) Second getmain base register
        LA    R10,2048(,R10) Second getmain base register
        LA    R6,2048(,R10) Third getmain base register
        LA    R6,2048(,R6)  Third getmain base register
        USING @STORE,R13,R10,R6  Getmain addressability
        STG   R2,@BACK      Save caller's save area pointer
        STG   R13,136(,R2)   Give caller our save area
        LR    R1,R3         Restore parameter register
@ENTRY2 EQU   * * * * * * * * End of the entry linkage code
        SPACE ,
PSEUDO  EQU   *           Dummy label used throughout

```

### BPX4ACC (access) example

The following code determines if file `/usr/inv/network.t` can be accessed. For the callable service, see [“access \(BPX1ACC, BPX4ACC\) — Determine if a file can be accessed” on page 24](#). For the data structure, see [“BPXYACC — Map flag values for access” on page 1055](#). AMODE 31 callers use [“BPX1ACC \(access\) example” on page 1219](#).

```

MVC    BUFFERA(18),=CL18'/usr/inv/network.t'
MVC    BUFLINA,=F'18'
XC     ACC(ACC#LENGTH),ACC
MVI    ACCINTENTFLAGS,ACC_R_OK+ACC_W_OK  Read and write access

```

## BPX4AC4 (access) example

```
SPACE ,
CALL BPX4ACC,          Determine accessibility of a file +
    (BUFLNA,          Input: Pathname length          +
    BUFFERA,          Input: Pathname                  +
    ACC,              Input: Access, BPXYACC            +
    RETVAL,           Return value: 0 or -1             +
    RETCODE,          Return code                      +
    RSNCODE),         Reason code                      +
    MF=(E,PLIST)      -----
SPACE ,
ICM  R15,B'1111',RETVAl  Set condition code for RETVAL
BZ   PSEUDO             Branch if RETVAL is zero
CLC  RETCODE,=A(EACCES)  Compare RETCODE to EACCES
BE   PSEUDO             Branch if access denied
```

## BPX4AC4 (accept4) example

The following code creates an **accept4** instance. For the callable service, use “accept4 (BPX1AC4, BPX4AC4) — Accept a connection request from a client socket and set flags” on page 16. AMODE 32 callers use the “BPX1AC4 (accept4) example” on page 1220.

```
CALL BPX4AC4,          Accept a socket connect request +
    (SOCKDESC,         Input: Socket descriptor+
    =A(SOCK#LEN+SOCK_SUN#LEN), Output: Write file descriptor +
    SOCADDR,           Input: Sockaddr structure +
    FLAGS,             Input: flag
    RETVAL,            Return value: 0 or -1 +
    RETCODE,           Return code +
    RSNCODE),          Reason code +
    VL,MF=(E,PLIST)
```

## BPX4ACK (auth\_check\_resource\_np) example

The following code determines if user 'JOEUSER' has UPDATE access to the FACILITY class profile 'TEST.THIS.PROFILE'. For the callable service, see “auth\_check\_resource\_np (BPX1ACK, BPX4ACK) — Determine a user’s access to a RACF-protected resource” on page 64. AMODE 31 callers use “BPX1ACK (auth\_check\_resource\_np) example” on page 1220.

```
MVI  CELLUUID,X'00'
MVI  PRNUUID,X'00'
MVC  USERNLEN,=F'7'
MVC  USERNAME(7),=CL7'JOEUSER'
MVC  CLSLEN,=F'8'
MVC  CLS(8),=CL8'FACILITY'
MVC  ENTLEN,=F'17'
MVC  ENT(17),=CL17'TEST.THIS.PROFILE'
SPACE ,
CALL BPX4ACK,          Determine access to a resource  +
    (CELLUUID,         Input: Cell UUID                +
    PRNUUID,           Input: Principal UUID            +
    USERNLEN,          Input: Userid length              +
    USERID,            Input: Userid                    +
    CLSLEN,,           Input: Class length               +
    CLS,               Input: Class                    +
    ENTLEN,            Input: Entity length              +
    ENT,               Input: Entity                    +
    =A(ACK_UPDATE#),   Input: Access type to check for  +
    RETVAL,            Return value: 0 or -1             +
    RETCODE,           Return code                      +
    RSNCODE),          Reason code                      +
    MF=(E,PLIST)      -----
```

## BPX4ACP (accept) example

The following code does an accept to accept a connect request from a client. SOCKDESC was previously set by a call to BPX4SOC. A bind and a listen must also have been previously done. The SOCKADDR was built by the call to BPX4BND. For the callable service, see “accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket” on page 13. For the data structure, see “BPXYSOCK — Map



SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1ACP (accept) example” on page 1220.

```

CALL  BPX4ACP,          Accept a socket connect request  +
      (SOCKDESC,        Input: Socket descriptor        +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr  +
      SOCKADDR,         Input: Sockaddr structure        +
      RETVAL,           Return value: 0 or -1            +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----
L  R2,RETVAL
ST  R2,SOCKDES2         Store the new socket descriptor

```

## BPX4AC4 (accept4) example

The following code creates an **accept4** instance. For the callable service, use “accept4 (BPX1AC4, BPX4AC4) — Accept a connection request from a client socket and set flags” on page 16. AMODE 32 callers use the “BPX1AC4 (accept4) example” on page 1220.

```

CALL  BPX4AC4,          Accept a socket connect request  +
      (SOCKDESC,        Input: Socket descriptor+
      =A(SOCK#LEN+SOCK_SUN#LEN), Output: Write file descriptor +
      SOCADDR,         Input: Sockaddr structure +
      FLAGS,           Input: flag
      RETVAL,          Return value: 0 or -1 +
      RETCODE,         Return code +
      RSNCODE),        Reason code +
      VL,MF=(E,PLIST)

```

## BPX4AIO (asyncio) example

The following code will accept the next conversation. For the callable service, see “asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets” on page 32. AMODE 31 callers use “BPX1AIO (asyncio) example” on page 1221.

```

XC  AIO(AIO#LENGTH),AIO  Null AIO control block
MVC AIOCMD,=A(AIO#ACCEPT) Command = Accept
MVC AIOFD,FILEDESC      File descriptor
MVC AIONOTIFYTYPE,=AL2(AIO#MVS) Notify type = MVS
XC  ECB01,ECB01         ECB = 0
LA  R15,ECB01           ECB Address
ST  R15,AIOECBPTR       Null AIO control block
MVC AIOSOCKADDRLEN,=A(SOCK#LEN)
LA  R15,SOCKADDR        From recvform (see BPX4RFM)
STG R15,AIOSOCKADDRPTR
SPACE
CALL BPX4AIO,           Asynchronous I/O for Sockets    +
      (=A(AIO#LENGTH), Input: Time before SIGAIOM      +
      AIO,              Input: Time before SIGAIOM      +
      RETVAL,           Return value: 0 or -1            +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4ALR (alarm) example

The following code schedules an alarm in 5 seconds. For the callable service, see “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 30. AMODE 31 callers use “BPX1ALR (alarm) example” on page 1221.

```

MVC  SECONDS,=F'5'
SPACE
CALL BPX4ALR,           Schedule Alarm                  +
      (SECONDS,         Input: Time before SIGALRM      +

```

RETVAL),	Return value: 0 or -1	+
MF=(E,PLIST)	-----	

## BPX4ANR (accept\_and\_rcv) example

The following code schedules an alarm in 5 seconds. For the callable service, see “alarm (BPX1ALR, BPX4ALR) — Set an alarm ” on page 30. AMODE 31 callers use “BPX1ALR (alarm) example” on page 1221.

MVC	SECONDS,=F'5'	
SPACE	,	
CALL	BPX4ALR,	Schedule Alarm
	(SECONDS,	Input: Time before SIGALRM
	RETVAL),	Return value: 0 or -1
	MF=(E,PLIST)	-----

## BPX4ASP (aio\_suspend) example

The following code will wait up to 10 seconds for one of the events specified in the AIOCB. For the callable service, see “aio\_suspend (BPX1ASP, BPX4ASP) — Wait for an asynchronous I/O request” on page 27. AMODE 31 callers use “BPX1ASP (aio\_suspend) example” on page 1222.

LA	R15,AIO	
STG	R15,ARGSLST	
MVC	ARGCNT,=F'1'	
MVC	SECONDS,=F'10'	
XC	NANOSECONDS,NANOSECONDS	
SPACE	,	
CALL	BPX4ASP,	Suspend for an aio request
	(ARGSLST,	Input: List of pointers to AIOCBs
	ARGCNT,	Input: Count of pointers in list
	SECONDS,	Input: Seconds to wait
	NANOSECONDS,	Input: Nanoseconds to wait
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4ATM (attach\_execmvs) example

The following code invokes program APPL92 on a subtask and as a child process of the caller, passing the length and parameter MONTH9,PRELIM,(232/74.99). There is no exit routine associated with program APPL92. For the callable service, see “attach\_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program ” on page 58. AMODE 31 callers use “BPX1ATM (attach\_execmvs) example” on page 1222.

MVC	PGMNAMEL,=F'6'	
MVC	PGMNAME(06),=CL6'APPL92'	
MVC	BUFLNA,=F'24'	
MVC	BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'	
SPACE	,	
CALL	BPX4ATM,	Invoke a MVS program
	(PGMNAMEL,	Input: Length of program name
	PGMNAME,	Input: Program name
	BUFLNA,	Input: Length of program argument
	BUFFERA,	Input: Program argument
	=AD(0),	Input: Exit routine address or 0
	=AD(0),	Input: Exit Parm list address or 0
	RETVAL,	Return value: Child PID Or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4ATX (attach\_exec) example

The program ictasma located at ict/bin gets control on a subtask and as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see “attach\_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program” on page 48. AMODE 31 callers use “BPX1ATX (attach\_exec) example” on page 1223.

```

MVC  BUFLINA,=F'16'
MVC  BUFFERA(16),=C'/ict/bin/ictasma'
MVC  ARGCNT,=F'3'

*
*      First
LA    R15,=F'4'          Length
STG   R15,ARGLIST+00     Length parm list
LA    R15,=CL4'WK18'     Argument
STG   R15,ARGSLIST+00    Argument address parm list

*
*      Second
LA    R15,=F'7'          Length
STG   R15,ARGLIST+08     Length parm list
LA    R15,=CL7'DEPT37A'  Argument
STG   R15,ARGSLIST+08    Argument address parm list

*
*      Third
LA    R15,=F'22'         Length
STG   R15,ARGLIST+16     Length parm list
LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)' Argument
STG   R15,ARGSLIST+16    Argument address parm list

*
MVC  ENVCNT,=F'0'        Number of env. data items passed
MVC  ENVLENS,=FD'0'      Addr of env. data length list
MVC  ENVPARMS,=FD'0'     Add of env. data

*
MVC  EXITRTNA,=AD(EXITRTN) ->exit routine
MVC  EXITPLA,=A(exit paramter list as expected by EXITRTN)

SPACE ,
CALL  BPX4ATX,
      (BUFLINA,          Input: Pathname length      +
      BUFFERA,           Input: Pathname              +
      ARGCNT,            Input: Argument count        +
      ARGLIST,           Input: Argument length list   +
      ARGSLIST,          Input: Argument address list  +
      ENVCNT,            Input: Environment count      +
      ENVLENS,           Input: Environment length list +
      ENVPARMS,          Input: Environment address list +
      EXITRTNA,          Input: Exit routine address or 0 +
      EXITPLA,           Input: Exit Parm list address or 0+
      RETVAL,            Return value: Child PID or -1  +
      RETCODE,           Return code                  +
      RSNCODE),          Reason code                  +
      MF=(E,PLIST)      -----

```

## BPX4AUT (\_\_authenticate) example

The following code authenticates a user with username and password and generates an identity token. For the callable service, use “\_\_authenticate (BPX1AUT, BPX4AUT) — Authenticate the specified user's credentials” on page 69. AMODE 31 callers use the “BPX1AUT (\_\_authenticate) example” on page 1223.

```

MVC  USERNLEN,=F'8'
MVC  USERNAME(8),=CL8'Myuserid'
MVC  OLDPASSLEN,=F'8'
MVC  OLDPASS(8),=CL8'MyOldPwd'
MVC  BUFLINA,=F'1024'
MVC  OPTIONS,=A(AUTH_BUILD_IDT#)

SPACE ,
CALL  BPX4AUT,
      (=A(AUTH_USER_ID#), authenticate users credentials +
      USERNLEN,          Input: Type of credentials      +
      USERNAME,          Input/Output: User ID length    +
      OLDPASSLEN,        Input: Pass_length              +
      OLDPASS,           Input: Pass                    +
      =A(0),             Input: New_pass_length          +
      NEWPASS,           Input: New_pass                 +
      BUFLINA,           Input: Idt_buffer_length        +
      BUFFERA,           Input: Idt_buffer_ptr           +
      IDTLEN,            Input/Output: Idt_length        +

```

## BPX4BND (bind) example

MSGBUFSP,	Input: Msg_buffer_subpool	+
MSGBUFPTR,	Output: Msg_buffer_ptr	+
=A(0),	Input: Appl_id_length	+
APPLID,	Input: Appl_id	+
OPTIONS,	Input/OUTPUT: Option_Flags	+
RETVAL,	Return value 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX4BND (bind) example

The following code does a bind to associate a name with a socket. SOCKDESC was previously set by a call to BPX4SOC. For the callable service, see “bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor” on page 76. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1BND (bind) example” on page 1224.

```
SPACE ,
MVI  SOCK_LEN,12          Store the length of the address
MVI  SOCK_FAMILY,AF_UNIX  Set the domain to AF_UNIX
MVC  SOCK_SUN_NAME(12),=CL12'/tmp/socket1' Set the name
CALL  BPX4BND,             Bind a name to a socket          +
      (SOCKDESC,           Input: Socket Descriptor        +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr  +
      SOCKADDR,            Input: Sockaddr structure       +
      RETVAL,              Return value: 0 or -1           +
      RETCODE,             Return code                     +
      RSNCODE),            Reason code                     +
      VL,MF=(E,PLIST)      -----
```

## BPX4BAS (bind with source address selection) example

The following code does a bind to associate the best source address for the provided destination IP address with a socket. SOCKDESC was previously set by a call to BPX4SOC. For the callable service, see “bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address” on page 79. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151.

```
SPACE ,
MVI  SOCK_LEN,=A(SOCK#LEN+SOCK_SIN6#LEN) Store the length of the address
MVI  SOCK_FAMILY,AF_INET6 Set the domain to AF_INET6
MVC  SOCK_SIN6_ADDR,=XL16'00A100B200C300D400E500F61234ABCD'
CALL  BPX4BAS,             Bind with source address selection+
      (SOCKDESC,           Input: Socket Descriptor        +
      =A(SOCK#LEN+SOCK_SIN6#LEN), Input:Length - Sockaddr  +
      SOCKADDR,            Input: Sockaddr structure       +
      RETVAL,              Return value: 0 or -1           +
      RETCODE,             Return code                     +
      RSNCODE),            Reason code                     +
      VL,MF=(E,PLIST)      -----
```

## BPX4CCA (cond\_cancel) example

The following code demonstrates how to cancel a program's interest in events that were selected by a call to the cond\_setup service. For the callable service, see “cond\_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events” on page 125. AMODE 31 callers use “BPX1CCA (cond\_cancel) example” on page 1225.

```
CALL  BPX4CCA,             Cancel cond_setup                +
      (RETVAL,             Return value: 0 or -1            +
      RETCODE,             Return code                     +
      RSNCODE),            Reason code                     +
      MF=(E,PLIST)        -----
```

- \* The return value (RETVAL) does not matter. When your program
- \* receives control following the call to cond\_cancel, it is no
- \* longer eligible to receive event notifications via cond\_post.

## BPX4CCS (\_\_console()) example

The following code sends a message to the console. For the callable service, see “[\\_\\_console\(\) \(BPX1CCS, BPX4CCS\) — Communicate with the console](#)” on page 142. For the data structure, see “[BPXYCCA — Map input/output structure for \\_\\_console\(\)](#)” on page 1060. AMODE 31 callers use “[BPX1CCS \(\\_\\_console\(\)\) example](#)” on page 1225.

```
CALL BPX4CCS,          Send msg to console          +
    (MSGATTRLEN,      Input: BPXYCCA length         +
    MSGATTR,          Input: BPXYCCA                +
    MODSTRINGPTR,     Output: Modify msg from console +
    MODIFYSTGLLEN,    Output: Length of modify msg   +
    CONMSGTYPE,       Output: Console msg type       +
    RETVAL,           Return value: 0 or -1          +
    RETCODE,          Return code                   +
    RSNCODE),         Reason code                   +
    MF=(E,PLIST)      -----
```

## BPX4CHA (chaudit) example

The following code changes the audit flags for the file identified by pathname. For the callable service, see “[chaudit \(BPX1CHA, BPX4CHA\) — Change audit flags for a file by path](#)” on page 95. For the data structure, see “[BPXYAUDT — Map flag values for chaudit and fchaudit](#)” on page 1060. AMODE 31 callers use “[BPX1CHA \(chaudit\) example](#)” on page 1225.

```
MVC  BUFFERA(18),=CL18'/usr/inv/network.t'
MVC  BUFLINA,=F'18'
MVI  AUDTREADACCESS,AUDTREADFAIL
MVI  AUDTWRITEACCESS,AUDTWRITEFAIL
MVI  AUDTEXECACCESS,AUDTEXECFAIL
MVI  AUDTRSRV,0
SPACE ,
CALL BPX4CHA,          Change audit                  +
    (BUFLINA,          Input: Pathname length         +
    BUFFERA,           Input: Pathname                +
    AUDT,              Input: Audit flags, BPXYAUDT   +
    =F'0',             Input: 0 user, 1 security auditor +
    RETVAL,            Return value: 0 or -1          +
    RETCODE,           Return code                   +
    RSNCODE),          Reason code                   +
    MF=(E,PLIST)      -----
```

## BPX4CHD (chdir) example

The following code changes the working directory for the task. For the callable service, see “[chdir \(BPX1CHD, BPX4CHD\) — Change the working directory](#)” on page 98. AMODE 31 callers use “[BPX1CHD \(chdir\) example](#)” on page 1226.

```
MVC  BUFFERA(8),=CL8'/usr/inv'
MVC  BUFLINA,=F'8'
SPACE ,
CALL BPX4CHD,          Change working directory      +
    (BUFLINA,          Input: Pathname length         +
    BUFFERA,           Input: Pathname                +
    RETVAL,            Return value: 0 or -1          +
    RETCODE,           Return code                   +
    RSNCODE),          Reason code                   +
    MF=(E,PLIST)      -----
```

## BPX4CHM (chmod) example

The following code changes the file mode for the file identified by pathname. For the callable service, see “[chmod \(BPX1CHM, BPX4CHM\) — Change the mode of a file or directory](#)” on page 101. For the data

## BPX4CHO (chown) example

structure, see “BPXYMODE — Map the mode constants ” on page 1106. AMODE 31 callers use “BPX1CHM (chmod) example” on page 1226.

```
MVC  BUFFERA(26),=CL26'newprogs/path/eightfold.c'
MVC  BUFLINA,=F'26'
XC   S_MODE,S_MODE
MVI  S_MODE2,S_IRUSR      All read and write
MVI  S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL BPX4CHM,             Change File Modes                +
      (BUFLINA,           Input: Pathname length            +
      BUFFERA,            Input: Pathname                    +
      S_MODE,             Input: Mode, mapped by BPXYMODE    +
      RETVAL,             Return value: 0 or -1              +
      RETCODE,            Return code                        +
      RSNCODE),           Reason code                        +
      MF=(E,PLIST)        -----
```

## BPX4CHO (chown) example

The following code changes the owner of /somedir/somefile.c from the current owner to that specified by USERID and GROUPID. For the callable service, see “chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory” on page 104. AMODE 31 callers use “BPX1CHO (chown) example” on page 1226.

```
MVC  BUFFERA(20),=CL20'/somedir/somefile.c'
MVC  BUFLINA,=F'20'
MVC  USERID,..           New owner UID from stat
MVC  GROUPID,..          New owner GID from stat
SPACE ,
CALL BPX4CHO,             Change owner and group of a file  +
      (BUFLINA,           Input: Pathname length            +
      BUFFERA,            Input: Pathname                    +
      USERID,             Input: New owner UID              +
      GROUPID,            Input: New owner GID              +
      RETVAL,             Return value: 0 or -1              +
      RETCODE,            Return code                        +
      RSNCODE),           Reason code                        +
      MF=(E,PLIST)        -----
```

## BPX4CHOA (fchownat) example

The following code changes ownership of a file relative to a directory file descriptor. For the callable service, use “fchownat (BPX1COA, BPX4COA) — Update the file ownership of a file or directory relative to a directory file descriptor ” on page 213. AMODE 31 callers use the “BPX1CHOA (fchownat) example” on page 1226.

```
MVC  FILEDESC,..         From opendir
MVC  GROUPID,..          Group ID 15
MVC  USERID,..          User ID 15
SPACE ,
CALL BPX4CHOA,           Change the owner and group of file+
      (FILEDESC,         Input: Directory File descriptor  +
      USERID,            Input: New user ID for file      +
      GROUPID,           Input: New group ID for file      +
      FLAGS,             Input: Flags                      +
      RETVAL,            Return value: 0 or -1            +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                      +
      VL,MF=(E,PLIST)    -----
```

## BPX4CHP (chpriority) example

The following code changes the CPU priority based on the input which, who, and priority type values. The which value used is PRIO\_PROCESS, indicating that the priority will be set by process ID. The who value used is 7, to set the priority for process ID 7. The priority type is CPRIO\_ABSOLUTE, indicating that the

priority will be set to the value specified, 1. For the callable service, see [“chpriority \(BPX1CHP, BPX4CHP\) — Change the scheduling priority of a process”](#) on page 107. AMODE 31 callers use [“BPX1CHP \(chpriority\) example”](#) on page 1227.

```

MVC PROCID,=XL4'00000007' Process ID to change priority for
MVC PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL BPX4CHP,          Change priority value          +
      (=A(PRIO_PROCESS), Input: Set by Process ID      +
      PROCID,          Input: PID to set priority for  +
      =A(CPRIO_ABSOLUTE), Input: Change by absolute value +
      PRIORITY,        Input: Priority value to change to+
      RETVAL,          Return value: 0 or -1          +
      RETCODE,         Return code                   +
      RSNCODE),        Reason code                   +
      MF=(E,PLIST)     -----
L    R15,RETVAL        Load return value
C    R15,=F'-1'        Test for -1 return
BE   PSEUDO            Branch on error

```

## BPX4CMA (fchmodat) example

The following code changes the file mode for the file identified by Pathname within the directory represented by the file descriptor FILEFD. Prior to this code excerpt, FILEFD would have been to the file descriptor of the specified directory or to AT\_FDCWD to indicate the current directory. For the callable service, see [“fchmodat \(BPX1CMA, BPX4CMA\) — Change the mode of a file or directory relative to a directory file descriptor”](#) on page 206. For the data structure, see [“BPXYMODE — Map the mode constants”](#) on page 1106. AMODE 31 callers use the [“BPX1CMA \(fchmodat\) example”](#) on page 1227.

```

*
*      Issue fchmodat
*
MVC  BUFLINA,=F'6'
MVC  BUFFERA(6),=CL6'test.c'
XC   S_MODE,S_MODE
XC   ATFLAGS,ATFLAGS
MVI  ATFLAGS,AT_SYMLINK_NOFOLLOW Do not follow symlinks
MVI  S_MODE2,S_IRUSR
MVI  S_MODE3,S_IRWXU2+S_IRGRP+S_IROTH
*
*      Permissions -
*      User All (RWX), Grp & Other Read
CALL BPX4CMA,          Change File Mode - fchmodat      +
      (BUFLINA,        Input: Pathname length          +
      BUFFERA,         Input: Pathname                  +
      S_MODE,          Input: Mode, mapped by BPXYMODE  +
      FILEDESC,        Input: Directory file desc       +
      ATFLAGS,         Input: At flags                  +
      RETVAL,          Return value: 0 or -1            +
      RETCODE,         Return code                     +
      RSNCODE),        Reason code                     +
      MF=(E,PLIST)
L    R15,RETVAL        Load return value
C    R15,=F'-1'        Test for -1 return
BE   PSEUDO            Branch on error
LR   R4,R15            Save the return value

```

## BPX4CHR (chattr) example

The following code changes the attributes of /somedir/somefile.c. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see [“chattr \(BPX1CHR, BPX4CHR\) — Change the attributes of a file or directory”](#) on page 82. For the data structures, see [“BPXYATT — Map](#)

## BPX4CLD (closedir) example

file attributes for `chattr` and `fchattr`” on page 1057 and “BPXYMODE — Map the mode constants ” on page 1106. AMODE 31 callers use “BPX1CHR (`chattr`) example” on page 1228.

```
MVC  BUFFERA(20),=CL20'/somedir/somefile.c'
MVC  BUFLINA,=F'20'
MVC  ATTID,=CL4'ATT '      Eye Catcher
MVC  ATTVERSION,=AL2(ATT#VER) version
XC   S_MODE,S_MODE        Clear mode
MVI  S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI  S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC  ATTMODE,S_MODE        Move mode data to attribute      +
      structure
MVC  ATTUID,=F'7'         Specify new UID
MVC  ATTGID,=F'77'        Specify new GID
OI   ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG      +
      Flag Mode, UID and GID changes
OI   ATTSETFLAGS2,ATTCTIMETOD                  +
      Set change time to current time
SPACE ,
CALL  BPX4CHR,              Change file attributes          +
      (BUFLINA,             Input: Pathname length         +
      BUFFERA,              Input: Pathname                +
      =A(ATT#LENGTH),       Input: BPXYATT length           +
      ATT,                  Input/output: BPXYATT           +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          -----
```

## BPX4CLD (closedir) example

The following code closes the directory identified by `FILEDESC`. For the callable service, see “`closedir` (BPX1CLD, BPX4CLD) — Close a directory” on page 124. AMODE 31 callers use “BPX1CLD (`closedir`) example” on page 1229.

```
MVC  FILEDESC,..          Directory descriptor from opendir
SPACE ,
CALL  BPX4CLD,             Close a directory                +
      (FILEDESC,           Input: Directory file descriptor +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                     +
      RSNCODE),            Reason code                     +
      MF=(E,PLIST)          -----
```

## BPX4CLN (clone) example

The following code clones a process that will be associated with a newly created mount namespace. The `CLNP` structure is filled in with the request information. The next sequential instruction obtains control from both the parent process (`RETVAL=child process ID`) and from the child process (`RETVAL=0`). If `RETVAL=-1`, the clone failed.

For the callable service, see `clone` (BPX1CLN, BPX4CLN) — Create a new process. For the data structure, see “BPXYCLNP— Map clone syscall parameters” on page 1062 and “BPXYSIGH — Signal constants” on page 1148. For the AMODE 31 example, see “BPX1CLN (clone) example” on page 1228.

```
LA   R5,CLN_PARM
USING CLNP,R5
XC   CLNP(CLNP#LEN),CLNP   Clear the input structure
MVC  CLNP_ID,=A(CLNP#ID)   CLNP identifier
MVHHi CLNP_VER,CLNP#VER    Current CLNP version
MVHHi CLNP_LEN,CLNP#LEN    CLNP length
MVC  CLNP_FLAGS,=A(CLONE_NEWNS) New mount namespace
MVC  CLNP_EXIT_SIGNAL,=A(SIGCHLD#) Must be SIGCHLD
CALL  BPX4CLN,              +
      (=A(CLNP#LEN),       Length of clone parameters    +
      CLN_PARM,            Clone parameters               +
      RETVAL,              Return value: -1, 0, child's PID +
      RETCODE,             Return code                     +
      RSNCODE),            Reason code                     +
      MF=(E,PLIST)
```



L	R15, RETVAL	Load return value
C	R15, =F'-1'	Test for -1 return
BE	PSEUDO	Branch on error

## BPX4CLO (close) example

The following code closes the standard input file. For the callable service, see [“close \(BPX1CLO, BPX4CLO\) — Close a file”](#) on page 121. AMODE 31 callers use [“BPX1CLO \(close\) example”](#) on page 1229.

CALL	BPX4CLO,	Close a file	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E, PLIST)	-----	

## BPX4CON (connect) example

The following code connects to a socket. SOCKDESC was returned by a previous call to BPX4SOC, and SOCKADDR contains the name of the peer, possibly obtained by a call to BPX4GNM. For the callable service, see [“connect \(BPX1CON, BPX4CON\) — Establish a connection between two sockets”](#) on page 139. For the data structure, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151. AMODE 31 callers use [“BPX1CON \(connect\) example”](#) on page 1229.

SPACE	,		
MVI	SOCK_LEN, 12	Store the length of the address	
MVI	SOCK_FAMILY, AF_UNIX	Set the domain to AF_UNIX	
MVC	SOCK_SUN_NAME(12), =CL12'/tmp/socket1'	Set the name	
CALL	BPX4CON,	Connect to a socket	+
	(SOCKDESC,	Input: Socket Descriptor	+
	=A(SOCK#LEN+SOCK_SUN#LEN),	Input: Length - Sockaddr	+
	SOCKADDR,	Input: Sockaddr structure	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E, PLIST)	-----	

## BPX4CPO (cond\_post) example

The following code demonstrates how to send an event notification to a thread waiting in the cond\_wait or cond\_timed\_wait service. For the callable service, see [“cond\\_post \(BPX1CPO, BPX4CPO\) — Post a thread for an event”](#) on page 127. AMODE 31 callers use [“BPX1CPO \(cond\\_post\) example”](#) on page 1230. The following code notifies thread (THID) that a CW\_CONDVAR event has occurred.

CALL	BPX4CPO,	Send condition event notification	+
	(THID,	Input: Thread ID of target pgm	+
	=A(CW_CONDVAR),	Input: Event in BPXYCW	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E, PLIST)	-----	

## BPX4CRA (chattrat) example

The following code modifies the attributes that are associated with a file. For the callable service, see [“chattrat \(BPX1CRA, BPX4CRA\) — Change the attributes of a file or directory relative to a directory file descriptor”](#) on page 91. For the data structures, see [“BPXYATT — Map file attributes for chattr and](#)

## BPX4CRT (chroot) example

fchattr” on page 1057 and “BPXYMODE — Map the mode constants ” on page 1106. AMODE 31 callers use “BPX1CRA (chatrat) example” on page 1230.

```
MVC
FILEDESC,..
File descriptor
MVC      BUFFERA(20),=CL20'/somedir/somefile.c'
MVC      BUFLINA,=F'20'
MVC      ATTID,=CL4'ATT
'
MVC      ATTVERSION,=AL2(ATT#VER)
XC        S_MODE,S_MODE
Clear mode
MVI      S_MODE2,S_IRUSR
Read-execute/write/read-execute
MVI      S_MODE3,S_IRUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC      ATTMODE,S_MODE
mode data to attribute + structure
MVC
ATTUID,=F'7'
Specify new UID
MVC
ATTGID,=F'77'
Specify new GID
OI
ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG
+
Flag Mode, UID and GID changes
OI
ATTSETFLAGS2,ATTCTIMETOD
+
Set change time to current time

SPACE ,
CALL BPX4CRA,
(FILEDESC,
BUFLINA,
BUFFERA,
=A(ATT#LENGTH),
ATT,
FLAGS,
RETVL,
RETCODE,
RSNCODE),
VL,MF=(E,PLIST)
```

Eye Catcher  
version

Move

Change file attributes +  
Input: File descriptor +  
Input: Pathname length +  
Input: Pathname +  
Input: BPXYATT length +  
Input/output: BPXYATT +  
Input: Flags +  
Return value: 0 or -1 +  
Return code +  
Reason code +  
-----

## BPX4CRT (chroot) example

The following code changes the root directory for the task. For the callable service, see “chroot (BPX1CRT, BPX4CRT) — Change the root directory” on page 111. AMODE 31 callers use “BPX1CRT (chroot) example” on page 1230.

```
MVC      BUFFERA(8),=CL8'/usr/inv'
MVC      BUFLINA,=F'8'
SPACE ,
CALL BPX4CRT,
(BUFLINA,
BUFFERA,
RETVL,
RETCODE,
RSNCODE),
MF=(E,PLIST)
```

Change root directory +  
Input: Pathname length +  
Input: Pathname +  
Return value: 0 or -1 +  
Return code +  
Reason code +  
-----

## BPX4CSE (cond\_setup) example

The following code sets up the invoker to suspend processing until any of the specified events (CW\_INTRPT or CW\_CONDVAR) occurs. The BPX4CTW (cond\_timed\_wait) or BPX4CWA (cond\_wait) service is used to actually suspend processing. For the callable service, see “cond\_setup (BPX1CSE,

BPX4CSE) — Set up to receive event notifications ” on page 130. AMODE 31 callers use “BPX1CSE (cond\_setup) example” on page 1231.

```

MVC  EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL  BPX4CSE,          Condition setup          +
      (EVENTLIST,      Input: Event list        BPXYCW +
      RETVAL,          Return value: 0 or -1      +
      RETCODE,         Return code              +
      RSNCODE),        Reason code              +
      MF=(E,PLIST)     -----

```

## BPX4CTW (cond\_timed\_wait) example

The following code suspends the calling thread until a signal arrives (CW\_INTRPT), or else 2.5 seconds have elapsed. For the callable service, see “cond\_timed\_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event ” on page 132. AMODE 31 callers use “BPX1CTW (cond\_timed\_wait) example” on page 1231.

```

MVC  EVENTLIST,=A(CW_INTRPT)          Signals
CALL  BPX4CTW,          Wait for condition events  +
      (=A(2),           Input: Number of seconds  +
      =A(500000000),    Input: Number of nanoseconds +
      EVENTLIST,        Input: Event list        BPXYCW +
      SECONDS,          Output: Unexpired seconds  +
      NANOSECONDS,      Output: Unexpired nanoseconds +
      RETVAL,           Return value: 0 or -1      +
      RETCODE,          Return code              +
      RSNCODE),         Reason code              +
      MF=(E,PLIST)     -----

```

## BPX4CWA (cond\_wait) example

The following code suspends the calling thread until either of two events occurs: the arrival of a signal (CW\_INTRPT) or some other thread using the cond\_post service to send this thread a CW\_CONDVAR notification. For the callable service, see “cond\_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event ” on page 136. AMODE 31 callers use “BPX1CWA (cond\_wait) example” on page 1231.

```

MVC  EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL  BPX4CWA,          Wait for condition events  +
      (EVENTLIST,      Input: Event list        BPXYCW +
      RETVAL,          Return value: 0 or -1      +
      RETCODE,         Return code              +
      RSNCODE),        Reason code              +
      MF=(E,PLIST)     -----

```

## BPX4DEL (deleteHFS) example

The program ictasma located at ict/bin is loaded into storage using BPX4LOD, branched to and then deleted from storage using BPX4DEL. For the callable service, see “deletehfs (BPX1DEL, BPX4DEL) — Delete a program from storage ” on page 148. AMODE 31 callers use “BPX1DEL (deleteHFS) example” on page 1232.

```

MVC  BUFLINA,=F'16'
MVC  BUFFERA(16),=C'/ict/bin/ictasma'
MVC  OPTIONS,=A(0)
MVC  LIBPTHLN,=A(0)
SPACE ,
CALL  BPX4LOD,          Load Program              +
      (BUFLINA,         Input: Pathname length    +
      BUFFERA,          Input: Pathname           +
      OPTIONS,          Input: Options            +
      LIBPTHLN,         Input: Library Path Length +
      LIBPATH,          Input: Library Path       +
      EPADDR,           Entry Point address       +

```

## BPX4EFD (eventfd) example

	RETVAL,	Return value	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	
SPACE	,		
LG	R15,EPADDR		
BASSM	R14,R15	Branch to loaded program	
SPACE	,		
CALL	BPX4DEL,	Delete program	+
	(EPADDR,	Input: Entry point address	+
	RETVAL,	Return value: -1 or 0	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4EFD (eventfd) example

The following code creates an **eventfd** instance. For the callable service, see “eventfd (BPX1EFD, BPX4EFD) — Create a file descriptor for event notification ” on page 160. AMODE 31 callers use the “BPX1EFD (eventfd) example” on page 1232.

MVI	INITVAL,1,	Set Initial count to 1	
XC	FLAGS(4),FLAGS		
MVI	FLAGS,EFD_SEMAPHORE,	READ with SEMAPHORE	
SPACE	,		
CALL	BPX4EFD,	Create a eventfd	+
	(INITVAL,	Input: Initial count	+
	FLAGS,	Input: eventfd Flags	BPXYEFD +
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BL	PSEUDO	Branch if negative (-1 = failure)	
ST	R15,FILEDESC	Store the file descriptor	

## BPX4ENV (oe\_env\_np) example

The following code enables interruption of threads waiting in MVS ENQs in the caller's process. For the callable service, see “oe\_env\_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute” on page 500. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 31 callers use “BPX1ENV (oe\_env\_np) example” on page 1233.

LA	R15,=F'1'		
ST	R15,INARG		
LA	R15,INARG		
STG	R15,INARGLIST		
LA	R15,INARGLIST		
STG	R15,INARGLISTPTR		
SPACE	,		
CALL	BPX4ENV,	oe_env_np	+
	(=A(ENQWAIT_PROCESS),	Input: Function_code	BPXYCONS +
	=A(1),	Input: InArgCount	+
	INARGLISTPTR,	Input: InArgListPtr	+
	=A(0),	Input: OutArgCount	+
	=AD(0),	Input: OutArgListPtr	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4EPC (epoll\_create) example

The following code creates an **epoll\_create** instance. For the callable service, use “epoll\_create (BPX1EPC, BPX4EPC) — Open an epoll file descriptor ” on page 150. AMODE 31 callers use the “BPX1EPC

(epoll\_create) example” on page 1233. For the data structure, see “BPXYEPOL— Map epoll syscall parameters” on page 1075.

```

XC      OPTION,OPTION
CALL    BPX4EPC,      Create an epoll file
          (OPTION,      Input: Option
          RETVAL,      Return value: EPOLL_FD or -1
          RETCODE,      Return code
          RSNCODE),    Reason code
          VL,MF=(E,PLIST)

```

## BPX4EPL (epoll\_epl ) example

The following code creates an **epoll\_ctl** instance. For the callable service, use “epoll\_ctl (BPX1EPL, BPX4EPL) — Control interface for epoll file descriptor” on page 152. AMODE 31 callers use the “BPX1EPL (epoll\_epl) example” on page 1233. For the data structure, see “BPXYEPOL— Map epoll syscall parameters” on page 1075.

```

XC      EPOLLEVENT(EPOLLEV#LEN),EPOLLEVENT
MVC     EPOLL_ID(8),=CL8'EPOLLEV '
MVI     EPOLLEVENTS,EPOLLIN+EPOLLOUT
MVC     EPOLL_FD(4),FILE_FD
CALL    BPX4EPL,      Control an epoll file
          (EPOLL_FD,      Input: Epoll instance from BPX1EPC
          EPOLL_CTL_ADD,  Input: Operation
          FILE_FD,      Input: file descriptor to monitor
          EPOLLEV#LEN,   Input: EpollEvent length
          EPOLLEVENT,    Input: EpollEvent Area
          RETVAL,      Return Value: 0 or -1
          RETCODE,      Return code
          RSNCODE),    Reason code
          VL,MF=(E,PLIST)

```

## BPX4EPW (epoll\_wait) example

The following code creates an **epoll\_wait** instance. For the callable service, use “epoll\_wait (BPX1EPW, BPX4EPW) — Wait for an I/O event on an epoll file descriptor” on page 157. AMODE 31 callers use the “BPX1EPW (epoll\_wait) example” on page 1234. For the data structure, see “BPXYEPOL— Map epoll syscall parameters” on page 1075.

```

LA      R15,BUFFERA
USING   EPOLLEVENT,R15
CALL    BPX4EPW,      Wait for I/O events
          (EPOLL_FD,      Input: Epoll instance from BPX1EPC
          MAXEVENTS,      Input: Max events fitting in EPOLLEVENT buffer
          TIMEOUT,      Input: How long to wait
          BUFFLEN,      Input: Length of buffer
          EPOLLEVENT,    Input: Epoll Events buffer
          RETVAL,      Return Value: -1 or Num Events returned
          RETCODE,      Return code
          RSNCODE),    Reason code
          VL,MF=(E,PLIST)

```

## BPX4EXC (exec) example

The program ictasma located at ict/bin gets control and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see “exec (BPX1EXC, BPX4EXC) — Run a program” on page 163. AMODE 31 callers use “BPX1EXC (exec) example” on page 1234.

```

MVC     BUFLINA,=F'16'
MVC     BUFFERA(16),=C'ict/bin/ictasma'
MVC     ARGCNT,=F'3'
*
First

```

## BPX4EXI (\_exit) example

```

      LA    R15,=F'4'          Length
      ST    R15,ARGLLST+00      Length parm list
      LA    R15,=CL4'WK18'      Argument
      STG   R15,ARGSLST+00      Argument address parm list
*
      LA    R15,=F'7'          Length
      ST    R15,ARGLLST+08      Length parm list
      LA    R15,=CL7'DEPT37A'   Argument
      STG   R15,ARGSLST+08      Argument address parm list
*
      LA    R15,=F'22'         Length
      ST    R15,ARGLLST+16      Length parm list
      LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)' Argument
      STG   R15,ARGSLST+16      Argument address parm list
*
      MVC   ENVCNT,=F'0'        Number of env. data items passed
      MVC   ENVLENS,=FD'0'      Addr of env. data length list
      MVC   ENVPARMS,=FD'0'     Add of env. data
*
      MVC   EXITRTNA,=AD(EXITRTN) ->exit routine
      MVC   EXITPLA,=AD(exit parameter list as expected by EXITRTN)
      SPACE
      CALL  BPX4EXC,              +
            (BUFLNA,              +
             BUFLNA,              +
             ARGCNT,              +
             ARGLLST,             +
             ARGSLST,             +
             ENVCNT,              +
             ENVLENS,             +
             ENVPARMS,            +
             EXITRTNA,            +
             EXITPLA,             +
             RETVAL,              +
             RETCODE,             +
             RSNCODE),            +
            MF=(E,PLIST)          +
            -----

```

## BPX4EXI (\_exit) example

The following code ends the program and returns an exit code of 44 to the waiting parent process. For the callable service, see “[\\_exit \(BPX1EXI, BPX4EXI\) — End a process and bypass the cleanup](#)” on page 180. AMODE 31 callers use “[BPX1EXI \(\\_exit\) example](#)” on page 1235.

```

      XC    WAST(WAST#LENGTH),WAST
      MVI   WASTEXITCODE,44      User defined exit code
      SPACE
      CALL  BPX4EXI,              +
            (WAST),              +
            MF=(E,PLIST)          +
            -----

```

## BPX4EXM (execmvs) example

The following code invokes program APPL92 and passes the length and parameter MONTH9,PRELIM, (232/74.99). There is no exit routine associated with program APPL92. For the callable service, see “[execmvs \(BPX1EXM, BPX4EXM\) — Run an MVS program](#)” on page 175. AMODE 31 callers use “[BPX1EXM \(execmvs\) example](#)” on page 1235.

```

      MVC   PGMNAMEL,=F'6'
      MVC   PGMNAME(06),=CL6'APPL92'
      MVC   BUFLNA,=F'24'
      MVC   BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
      SPACE
      CALL  BPX4EXM,              +
            (PGMNAMEL,            +
             PGMNAME,             +
             BUFLNA,              +
             BUFFERA,             +
             =AD(0),              +
             =AD(0),              +
             RETVAL,              +
             RETCODE),            +
            Invoke a MVS program  +
            Input: Length of program name  +
            Input: Program name            +
            Input: Length of program argument +
            Input: Program argument        +
            Input: Exit routine address or 0 +
            Input: Exit Parm list address or 0+
            Return value: -1 or not return  +
            Return code                    +
            -----

```

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4EXT (extlink\_np) example

The following code creates an external link to data set MY.DATASET for pathname /mvs/mydataset. For the callable service, see “[extlink\\_np \(BPX1EXT, BPX4EXT\) — Create an external symbolic link](#)” on page 183. AMODE 31 callers use “[BPX1EXT \(extlink\\_np\) example](#)” on page 1235.

MVC	BUFFERA(10),=CL10'MY.DATASET'		
MVC	BUFLINA,=F'10'		
MVC	BUFFERB(14),=CL14'/mvs/mydataset'		
MVC	BUFLINB,=F'14'		
SPACE	,		
CALL	BPX4EXT,	Create external link to name	+
	(BUFLINA,	Input: External name length	+
	BUFFERA,	Input: External name	+
	BUFLINB,	Input: Link name length	+
	BUFFERB,	Input: Link name	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4FAA (faccessat) example

The following code creates the file for the file identified by pathname within the directory represented by the file descriptor FILEFD. Previously, FILEFD would have been to the file descriptor of the desired directory or to AT\_FDCWD to indicate the current directory. For the callable service, see “[faccessat \(BPX1FAA, BPX4FAA\) — Checks user's permissions of a file relative to a directory file descriptor](#)” on page 186. AMODE 31 callers use “[BPX1FAA \(faccessat\) example](#)” on page 1235.

MVC	BUFFERA(18),=CL18'/usr/inv/network.t'		
MVC	BUFLINA,=F'18'		
XC	ACC(ACC#LENGTH),ACC		
MVI	ACCINTENTFLAGS,ACC_R_OK+ACC_W_OK	Read and write access	
XC	ATFLAGS,ATFLAGS		
MVI	ATFLAGS,AT_EACCESS	Use effective user ID	
SPACE	,		
CALL	BPX4FAA,	Determine accessibility of a file	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	ACC,	Input: Access, BPXYACC	+
	FILEDESC,	Input: Directory file descriptor	+
	ATFLAGS,	Input: At flags	
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	
SPACE	,		
ICM	R15,B'1111',RETVAL	Set condition code for RETVAL	
BZ	PSEUDO	Branch if RETVAL is zero	
CLC	RETCODE,=A(EACCES)	Compare RETCODE to EACCES	
BE	PSEUDO	Branch if access denied	

## BPX4FAI (freeaddrinfo) example

The following code frees the Addr\_Info structure(s) that were obtained by the getaddrinfo callable service. For the callable service, see “[freeaddrinfo \(BPX1FAI, BPX4FAI\) — Free Addr\\_Info structures](#)” on page 236. AMODE 31 callers use “[BPX1FAI \(freeaddrinfo\) example](#)” on page 1236.

SPACE	,		
CALL	BPX4FAI,	Free Addr_Info	+
	(ADDR_INFO_PTR,	Input: -> Addr_Info structure	+
	RETVAL,	Return code	+
	RETCODE,	Return code	+

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4FCA (fchaudit) example

The following code changes the audit for the standard input file to ReadFail, WriteFail and ExecFail. For the callable service, see [“fchaudit \(BPX1FCA, BPX4FCA\) — Change audit flags for a file by descriptor”](#) on page 199. For the data structure, see [“BPXYAUDT — Map flag values for chaudit and fchaudit”](#) on page 1060. AMODE 31 callers use [“BPX1FCA \(fchaudit\) example”](#) on page 1236.

MVI	AUDTREADACCESS,AUDTREADFAIL	
MVI	AUDTWRITEACCESS,AUDTWRITEFAIL	
MVI	AUDTEXECACCESS,AUDTEXECFAIL	
MVI	AUDTRSRV,X'00'	
SPACE	,	
CALL	BPX4FCA,	Change audit
	(=A(STDIN_FILENO),	Input: File descriptor
	AUDT,	Input: Audit flags, BPXYAUDT
	=A(0),	Input: 0 user, 1 security auditor
	RETVL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4FCD (fchdir) example

The following code changes the working directory for the task to the directory identified by FILEDESC. For the callable service, see [“fchdir \(BPX1FCD, BPX4FCD\) — Change the working directory”](#) on page 202. AMODE 31 callers use [“BPX1FCD \(fchdir\) example”](#) on page 1237.

MVC	FILEDESC,..	Directory descriptor from opendir
SPACE	,	
CALL	BPX4FCD,	Change working directory
	(FILEDESC,	Input: Directory file descriptor
	RETVL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4FCM (fchmod) example

The following code changes the permissions for the standard input file. For the callable service, see [“fchmod \(BPX1FCM, BPX4FCM\) — Change the mode of a file or directory by descriptor”](#) on page 204. For the data structure, see [“BPXYMODE — Map the mode constants”](#) on page 1106 and [“BPXYFTYP — File type definitions”](#) on page 1078. AMODE 31 callers use [“BPX1FCM \(fchmod\) example”](#) on page 1237.

XC	S_MODE,S_MODE	
MVI	S_MODE2,S_IRUSR	All permissions
MVI	S_MODE3,S_IRWXU2+S_IRWXG+S_IRWXO	
SPACE	,	
CALL	BPX4FCM,	Change file modes
	(=A(STDIN_FILENO),	Input: File descriptor
	S_MODE,	Input: Mode, BPXYMODE, BPXYFTYP
	RETVL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4FCO (fchown) example

The following code changes the owner and group for the standard input file. For the callable service, see [“fchown \(BPX1FCO, BPX4FCO\) — Change the owner and group of a file or directory by descriptor”](#) on page



210. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 31 callers use “BPX1FCO (fchown) example” on page 1237.

MVC	GROUPID,..	Group ID
MVC	USERID,..	User ID
SPACE ,		
CALL	BPX4FCO,	Change the owner and group of file+
	(=A(STDIN_FILENO),	Input: File descriptor +
	USERID,	Input: New user ID for file +
	GROUPID,	Input: New group ID for file +
	RETVAL,	Return value: 0 or -1 +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	MF=(E,PLIST)	-----

## BPX4FCR (fchattr) example

The following code changes the attributes of the standard input file. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see “fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor” on page 190. For the data structures, see “BPXYATT — Map file attributes for chattr and fchattr” on page 1057 and “BPXYMODE — Map the mode constants ” on page 1106. AMODE 31 callers use “BPX1FCR (fchattr) example” on page 1237.

MVC	FILEDESC,..	File descriptor
MVC	BUFFERA(20),=CL20'/somedir/somefile.c'	
MVC	BUFLINA,=F'20'	
MVC	ATTID,=CL4'ATT '	Eye Catcher
MVC	ATTVERSION,=AL2(ATT#VER)	version
XC	S_MODE,S_MODE	Clear mode
MVI	S_MODE2,S_IRUSR	Read-execute/write/read-execute
MVI	S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH	
MVC	ATTMODE,S_MODE	Move mode data to attribute + structure
MVC	ATTUID,=F'7'	Specify new UID
MVC	ATTGID,=F'77'	Specify new GID
OI	ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG	
+		
		Flag Mode, UID and GID changes
OI	ATTSETFLAGS2,ATTCTIMETOD	
+		
		Set change time to current time
SPACE ,		
CALL	BPX4CRA,	Change file attributes +
	(FILEDESC,	Input: File descriptor +
	BUFLINA,	Input: Pathname length +
	BUFFERA,	Input: Pathname +
	=A(ATT#LENGTH),	Input: BPXYATT length +
	ATT,	Input/output: BPXYATT +
	FLAGS,	Input: Flags +
	RETVAL,	Return value: 0 or -1 +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	VL,MF=(E,PLIST)	-----

## BPX4FCT (fcntl) example

The code for the first example duplicates the standard error file descriptor to a file descriptor greater than or equal to FILEDES2.

The code for the second example sets a shared byte range lock. For the callable service, see “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 216. For the data structure, see “BPXYFCTL — Command values and flags for fcntl” on page 1076, “BPXYBRLK — Map byte range lock request for fcntl” on page 1060, and “BPXYOPNF — Map flag values for open” on page 1113. AMODE 31 callers use “BPX1FCT (fcntl) example” on page 1238.

* for 2nd parm F_DUPFD, F_DUPFD2	3rd parm file desc no..
* for 2nd parm F_GETFD, F_GETFL	3rd parm 0

## BPX4FLK (flock) example

```
* for 2nd parm F_SETFD          3rd parm BPXYFCTL
* for 2nd parm F_GETLK, F_SETLK, F_SETLKW 3rd parm BPXYBRLK
* for 2nd parm F_SETFL          3rd parm BPXYOPNF
SPACE ,
* Example 1 - duplicate file descriptor
MVC FILEDES2,=F'20'          Get free file descriptor >= 20
SPACE ,
CALL BPX4FCT,                General purpose file control      +
(=A(STDERR_FILENO),          Input: File descriptor          +
=A(F_DUPFD),                 Input: Action, BPXYFCTL      +
FILEDES2,                    Input: Argument #/0/FCTL/BRLK/OPNF+
RETVAL,                       Return value: 0, -1 or action  +
RETCODE,                       Return code                +
RSNCODE),                     Reason code                  +
MF=(E,PLIST)                  -----
SPACE ,
* Example 2 - duplicate file descriptor
MVC FILEDES2,=F'20'          Get next higher file descriptor
LA R15,BRLK
STG R15,BRLKA
XC BRLK(BRLK#LENGTH),BRLK    Null out BRLK
MVC L_TYPE,=AL2(F_RDLCK)      Lock type = shared
MVC L_WHENCE,=AL2(SEEK_CUR)    Whence = from current cursor
SPACE ,
CALL BPX4FCT,                General purpose file control      +
(=A(STDERR_FILENO),          Input: File descriptor          +
=A(F_SETLK),                 Input: Action, BPXYFCTL      +
BRLKA,                       Input: Argument #/0/FCTL/BRLK/OPNF+
RETVAL,                       Return value: 0, -1 or action  +
RETCODE,                       Return code                +
RSNCODE),                     Reason code                  +
MF=(E,PLIST)                  -----
```

## BPX4FLK (flock) example

The following code serializes open files in the z/OS UNIX file system. For the callable service, use “[flock \(BPX1FLK, BPX4FLK\) — Apply or remove an advisory lock on an open file](#)” on page 225. AMODE 31 callers use the “[BPX1FLK \(flock\) example](#)” on page 1239.

```
LHI      @08,14
ST        @08,ACTIONCODE
CALL BPX4FLK(testFileFD,ActionCode,Rv,Rc,Rs);
L         @05,RV
CHI       @05,-1
JNE       @RF00101
```

## BPX4FPC (fpathconf) example

The following code obtains the configurable option associated with the pipe buffer. For the callable service, see “[fpathconf \(BPX1FPC, BPX4FPC\) — Determine configurable path name variables](#)” on page 233. For the data structure, see “[BPXYPCF — Command values for pathconf and pathconf](#)” on page 1114. AMODE 31 callers use “[BPX1FPC \(fpathconf\) example](#)” on page 1239.

```
MVC FILEDESC,..              From opendir
SPACE ,
CALL BPX4FPC,                Get configurable pathname variable+
(FILEDESC,                   Input: Directory file descriptor +
=A(PC_PIPE_BUF),             Input: Configurables BPXYPCF +
RETVAL,                       Return value: 0, -1 or variable  +
RETCODE,                       Return code                +
RSNCODE),                     Reason code                  +
MF=(E,PLIST)                  -----
```

## BPX4FRK (fork) example

The following code forks a new process. The next sequential instruction gets control from both the parent process (RETVAL=child process ID) and from the child process (RETVAL=0). If RETVAL=-1, the fork failed.

For the callable service, see [“fork \(BPX1FRK, BPX4FRK\) — Create a new process ” on page 227](#). AMODE 31 callers use [“BPX1FRK \(fork\) example” on page 1239](#).

CALL	BPX4FRK,	Create a new process (fork)	+
	(RETVAL,	Return value: -1, 0, child's PID	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4FST (fstat) example

The following code gets the file status for the file opened as FILEDESC. For the callable service, see [“fstat \(BPX1FST, BPX4FST\) — Get status information about a file by descriptor ” on page 238](#). For the data structure, see [“BPXYSTAT — Map the response structure for stat ” on page 1162](#). AMODE 31 callers use [“BPX1FST \(fstat\) example” on page 1239](#).

MVC	FILEDESC,..	File descriptor from open	
	SPACE		
CALL	BPX4FST,	Get file status of file descriptor	+
	(FILEDESC,	Input: File descriptor	+
	STATL,	Input: Length of buffer needed	+
	STAT,	Buffer, mapped by BPXYSTAT	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX1FSA (fstatat) example

The following code gets the file status of the file that was specified by the path BUFFERA in the directory specified by the directory descriptor DIRDESC. For the callable service, see [“fstatat \(BPX1FSA, BPX4FSA\) — Get file status relative to a directory file descriptor ” on page 241](#). For the data structure, see [“BPXYSTAT — Map the response structure for stat ” on page 1162](#). AMODE 31 callers use [“BPX1FSA \(fstatat\) example” on page 1240](#),

```

MVC    DIRDESC      Directory Descriptor from OPENDIR
MVC    FLAGS,F='0'   Defined in BPXYFCTL
MVC    BUFLINA,F='6'
MVC    BUFFERA(6),=CL6'file.t'
SPACE
CALL   BPX4FST      Get File Status
      (BUFLINA,      Input: Path length
      BUFFERA,       Input: Path
      STATL,         Stat length
      STAT,          Stat buffer
      DIRDESC,       Input: Directory descriptor
      Flags,         Input: Flags
      RETVAL         Return value: 0 or -1
      RETCODE        Return code
      RSNCODE)       Reason code
      VL,MF=(E,PLIST)

```

## BPX4FSY (fsync) example

The following code writes file descriptor changes to permanent storage. For the callable service, see [“fsync \(BPX1FSY, BPX4FSY\) — Write changes to permanent storage” on page 247](#). AMODE 31 callers use [“BPX1FSY \(fsync\) example” on page 1240](#).

MVC	FILEDESC,..	File descriptor from open	
	SPACE		
CALL	BPX4FSY,	Write changes to permanent storage	+
	(FILEDESC,	Input: File descriptor	+

## BPX4FTR (ftruncate) example

RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4FTR (ftruncate) example

The following code truncates the file described by FILEDESC after 512 bytes. For the callable service, see [“ftruncate \(BPX1FTR, BPX4FTR\) — Change the size of a file” on page 249](#). AMODE 31 callers use [“BPX1FTR \(ftruncate\) example” on page 1240](#).

MVC	FILEDESC,..	File descriptor from open	
MVC	NEWLEN(8),=FL8'512'		
SPACE	,		
CALL	BPX4FTR,	Truncate a file	+
	(FILEDESC,	Input: File descriptor	+
	NEWLEN,	Input: Length to keep	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4FTV (fstatvfs) example

The following code obtains information about the file system containing the file identified by FILEDESC. For the callable service, see [“fstatvfs \(BPX1FTV, BPX4FTV\) — Get the file system status” on page 244](#). For the data structure, see [“BPXYSSTF — Map response structure for file system status” on page 1161](#). AMODE 31 callers use [“BPX1FTV \(fstatvfs\) example” on page 1240](#).

MVC	FILEDESC,..	File descriptor from open	
SPACE	,		
CALL	BPX4FTV,	Get file system status	+
	(FILEDESC,	Input: File descriptor	+
	SSTFL,	Input: Length of BPXYSSTF	+
	SSTF,	Buffer, BPXYSSTF	+
	RETVAL,	Return value: -1 or length status	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4GAI (getaddrinfo) example

The following code returns the IP address and other associated information for the specified node name. For the callable service, see [“getaddrinfo \(BPX1GAI, BPX4GAI\) — Get the IP address and information for a service name or location” on page 251](#). AMODE 31 callers use [“BPX1GAI \(getaddrinfo\) example ” on page 1241](#).

SPACE	,		
CALL	BPX4GAI,	Get Addr_info	+
	(NODE_NAME,	Input: Name of Host being queried	+
	NODE_NAME_LENGTH,	Input: Length of host name	+
	SERVICE_NAME,	Input: Service name being queried	+
	SERVICE_NAME_LENGTH,	Input: Length of service name	+
	HINTS_PTR,	Input: Ptr to Addr_Info Structure	+
	RESULTS_PTR,	Output:Ptr to Addr_Info Structure	+
	CANONICAL_LENGTH,	Output: Length canonical name	+
	RETVAL,	Return code	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4GCL (getclientid) example

The following code obtains the clientid information for caller. This information is used on givesocket (BPX4GIV) and takesocket (BPX4TAK) services. For the callable service, see “getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier” on page 258. For the data structure, see “BPXYCID — Map the returning structure for getclientid()” on page 1061. AMODE 31 callers use “BPX1GCL (getclientid) example” on page 1241.

CALL	BPX4GCL,	get clientid information	+
	(=F'2',	Input: Function code of 2	+
	=A(AF_INET),	Input: Domain of AF_INET	+
	CID,	Output: Clientid information	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4GCW (getcwd) example

The following code gets the working directory for the caller. For the callable service, see “getcwd (BPX1GCW, BPX4GCW) — Get the path name of the working directory” on page 260. AMODE 31 callers use “BPX1GCW (getcwd) example” on page 1241.

MVC	BUFLINA,=F'1024'	Max directory name return area	
SPACE	,		
CALL	BPX4GCW,	Get working directory name	+
	(BUFLINA,	Input: Length directory work area	+
	BUFFERA,	Buffer	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4GEG (getegid) example

The following code gets the effective group ID of the caller. For the callable service, see “getegid (BPX1GEG, BPX4GEG) — Get the effective group ID” on page 263. AMODE 31 callers use “BPX1GEG (getegid) example” on page 1242.

CALL	BPX4GEG,	Get the effective group ID	+
	(RETVAL),	Return value: effective group ID	+
	MF=(E,PLIST)	-----	

## BPX4GEP (getpgid) example

The following code returns the process group ID for the process identified by the input process ID. The process ID value is set to 1. For the callable service, see “getpgid (BPX1GEP, BPX4GEP) — Get the process group ID” on page 295. AMODE 31 callers use “BPX1GEP (getpgid) example” on page 1242.

MVC	PROCID,=XL4'00000001'	Value of process ID	
SPACE	,		
CALL	BPX4GEP,	Get process group ID	+
	(PROCID,	Input: Process ID	+
	RETVAL,	Return value: process group ID	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX4GES (getsid) example

The following code returns the process group ID for the session leader of the process identified by the input process ID. The process ID value is set to 1. For the callable service, see [“getsid \(BPX1GES, BPX4GES\) – Get the process group ID of the session leader”](#) on page 317. AMODE 31 callers use [“BPX1GES \(getsid\) example”](#) on page 1242.

MVC	PROCID,=XL4'00000000'	Value of process ID	
SPACE	,		
CALL	BPX4GES,	Get group ID of session leader	+
	(PROCID,	Input: Process ID	+
	RETVAL,	Return value: process group ID	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX4GET (w\_getipc) example

The following code retrieves information on the first semaphore defined to the system to which the caller has read access. For the callable service, see [“w\\_getipc \(BPX1GET, BPX4GET\) – Query interprocess communications”](#) on page 991. For the data structure, see [“BPXYIPCQ – Map w\\_getipc structure”](#) on page 1098. AMODE 31 callers use [“BPX1GET \(w\\_getipc\) example”](#) on page 1242.

XC	TOKEN,TOKEN	Zero, token for 1st member	
LA	R5,BUFFERA	Area for query IPC return data	
STG	R5,BUFA	R5 -> IPCQ	
SPACE	,		
CALL	BPX4GET,	Interprocess Communications	+
	(TOKEN,	Input: member token	+
	BUFA,	Input: ->IPCQ	BPXYIPCQ+
	=A(IPCQ#LENGTH),	Input: Length of IPCQ	BPXYIPCQ+
	=A(IPCQ#SEM),	Input: Request	BPXYIPCQ+
	RETVAL,	Return value: 0, -1 or value	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
SPACE	,		
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	
LTR	R15,R15	Test for 0 return	
BZ	PSEUDO	Branch on end of file	
ST	R15,TOKEN	Save token for next w_semipc	

## BPX4GEU (geteuid) example

The following code gets the effective user ID of the caller. For the callable service, see [“geteuid \(BPX1GEU, BPX4GEU\) – Get the effective user ID”](#) on page 264. AMODE 31 callers use [“BPX1GEU \(geteuid\) example”](#) on page 1243.

CALL	BPX4GEU,	Get the effective user ID	+
	(RETVAL),	Return value: effective user ID	+
	MF=(E,PLIST)	-----	

## BPX4GGE (getgrent) example

The following code accesses the group database by starting with the next available entry and continuing until the end of file on the database. It returns a structure that identifies information about each group entry in the database. For the callable service, see [“getgrent \(BPX1GGE, BPX4GGE\) – Sequentially access the group database”](#) on page 266. For the data structure, see [“BPXYGIDS – Map data returned for](#)

getgrnam and getgrpid” on page 1080. AMODE 31 callers use “BPX1GGE (getgrent) example” on page 1243.

```

GGELoop DS 0H
CALL BPX4GGE,      Access the group database      +
      (RETVAL,      Return value: 0 or ->BPXYGIDS    +
      RETCODE,      Return code                    +
      RSNCODE),     Reason code                    +
      MF=(E,PLIST)  -----
      ICM R8,B'1111',RETVAL
      BZ  CHKGGERR   Error or end of file
      USING GIDS,R8
*      access the group structure
      DROP R8
      B  GGELoop    Check next group entry
CHKGGERR DS 0H
      ICM R8,B'1111',RETCODE
      BZ  GGEEOF     End of file
*      handle error as needed
GGEEOF  DS 0H

```

## BPX4GGI (getgrgid) example

The following code accesses the group database by the ID of the caller and returns a structure identifying the groups by ID. The group ID value is set to 5. For the callable service, see “getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID” on page 268. For the data structure, see “BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 1080. AMODE 31 callers use “BPX1GGI (getgrgid) example” on page 1243.

```

MVC GROUPID,=XL4'00000005' Value of group ID
SPACE ,
CALL BPX4GGI,      Access the group database      +
      (GROUPID,      Input: Group ID                +
      RETVAL,      Return value: 0 or ->BPXYGIDS    +
      RETCODE,      Return code                    +
      RSNCODE),     Reason code                    +
      MF=(E,PLIST)  -----
      ICM R8,B'1111',RETVAL
      BZ  NOGIDS
      USING GIDS,R8
*      access the group structure
      DROP R8
NOGIDS EQU *

```

## BPX4GGN (getgrnam) example

The following code accesses the group database by the name of the caller and returns a structure identifying the groups by ID. For the callable service, see “getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name” on page 271. For the data structure, see “BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 1080. AMODE 31 callers use “BPX1GGN (getgrnam) example” on page 1244.

```

MVC GRNAMELN,=F'7'
MVC GRPGMNAME(7),=CL7'EXTSERV'
SPACE ,
CALL BPX4GGN,      Access the group database      +
      (GRNAMELN,      Input: Length of group name    +
      GRPGMNAME,      Input: Name of group           +
      RETVAL,      Return value: 0 or ->BPXYGIDS    +
      RETCODE,      Return code                    +
      RSNCODE),     Reason code                    +
      MF=(E,PLIST)  -----

```

## BPX4GGR (getgroups) example

The following code provides the caller with a list of supplementary group IDs. The code sets BUFW size to 256. The actual BUFW size is determined from the previous BPX4GGR RETVAL when BUFW was 0. For the callable service, see [“getgroups \(BPX1GGR, BPX4GGR\) – Get a list of supplementary group IDs” on page 274](#). AMODE 31 callers use [“BPX1GGR \(getgroups\) example” on page 1244](#).

```

*      MVC   BUFW,=XL4'00000256'   Value of buffer BUFW
      LA     R15,BUFFERA           Space for BUFW words
      STG    R15,BUFA              ->Array for group IDs
      SPACE ,
      CALL   BPX4GGR,              Get list of supplementary grp IDs +
      (BUFW,                        Input: Group ID list size      +
      BUFA,                        ->Buffer for Group ID list address+
      RETVAL,                      Return value: -1, 0, ID count    +
      RETCODE,                     Return code                     +
      RSNCODE),                   Reason code                       +
      MF=(E,PLIST)                -----

```

## BPX4GHA (gethostbyaddr) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose address is specified as input. For the callable service, see [“gethostbyaddr \(BPX1GHA, BPX4GHA\) Get the IP address and alias of a host name for the specified IP address” on page 279](#).

The HOSTENT structure has the following format:

- `h_name` - The address of the host name returned by the service. The host name is a variable length field that is ended by `x'00'`.
- `h_aliases` - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer `x'00000000'`. Each alias name is a variable length field that is ended by `x'00'`.
- `h_addrtype` - The value 2, which signifies `AF_INET`.
- `h_length` - The length of the host internet addresses pointed to by `h_addr_list`.
- `h_addr_list` - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer `x'00000000'`.

AMODE 31 callers use [“BPX1GHA \(gethostbyaddr\) example” on page 1244](#).

```

*      MVC   HOST_ADDR,=XL4'C90E0256'   IP Address of Host
      MVC   HOST_ADDRLN,=F'4'           Address length
      SPACE ,
      CALL   BPX4GHA,                  Get host by address          +
      (HOST_ADDR,                      Input: IP address of queried HOST +
      HOST_ADDRLN,                     Input: Length of IP address      +
      HOSTENT_PTR,                     Output: 0 or -> HOSTENT structure +
      =A(AF_INET),                     Input: Domain - AF_INET          +
      RETVAL,                          Return code                     +
      RETCODE,                         Return code                     +
      RSNCODE),                       Reason code                       +
      MF=(E,PLIST)                    -----

```

## BPX4GHN (gethostbyname) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose domain name is specified as input. For the callable service, see [“gethostbyname \(BPX1GHN, BPX4GHN\) Get IP information for specified host domain names” on page 282](#).

The HOSTENT structure has the following format:

- `h_name` - The address of the host name returned by the service. The host name is a variable length field that is ended by `x'00'`.



- h\_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h\_addrtype - The value 2, which signifies AF\_INET.
- h\_length - The length of the host internet addresses pointed to by h\_addr\_list.
- h\_addr\_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

AMODE 31 callers use [“BPX1GHN \(gethostbyname\) example”](#) on page 1245.

```

MVC  HOST_NAME(8),=CL8'HOST1234'
MVC  HOST_NAMELEN,=F'8'
SPACE ,
CALL  BPX4GHN,           Get host by name          +
      (HOST_NAME,       Input: Name of Host being queried +
      HOST_NAMELEN,     Input: Length of host name      +
      HOSTENT_PTR,      Output: 0 or -> HOSTENT structure +
      RETVAL,           Return code                    +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4GID (getgid) example

The following code gets the real group ID of the caller. For the callable service, see [“getgid \(BPX1GID, BPX4GID\) — Get the real group ID”](#) on page 265. AMODE 31 callers use [“BPX1GID \(getgid\) example”](#) on page 1245.

```

CALL  BPX4GID,           Get the real group ID          +
      (RETVAL),          Return value: real group ID    +
      MF=(E,PLIST)      -----

```

## BPX4GIV (givesocket) example

The following code gives a socket to the program identified by CID (clientid). The target program may then use takesocket (BPX4TAK) to take the socket. SOCKDESC was previously set by a call to BPX4ACP. CID is set by the getclientid (BPX4GCL) service. For the callable service, see [“givesocket \(BPX1GIV, BPX4GIV\) — Give a socket to another program”](#) on page 332. For the data structure, see [“BPXYCID — Map the returning structure for getclientid\(\)”](#) on page 1061. AMODE 31 callers use [“BPX1GIV \(givesocket\) example”](#) on page 1245.

```

CALL  BPX4GIV,           give a socket to another program +
      (SOCKDESC,         Input: Socket descriptor        +
      CID,               Input: Clientid of recipient    +
      RETVAL,            Return value: 0 or -1           +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4GLG (getlogin) example

The following code gets the login name of the caller. For the callable service, see [“getlogin \(BPX1GLG, BPX4GLG\) — Get the user login name”](#) on page 289. AMODE 31 callers use [“BPX1GLG \(getlogin\) example”](#) on page 1246.

```

CALL  BPX4GLG,           Get the login name            +
      (RETVAL),          Returns value, 0 or ->login name +
      MF=(E,PLIST)      -----

```

## BPX4GMN (w\_getmntent) example

The following code gets the mount entries for the caller. For the callable service, see “w\_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems ” on page 995. For the data structure, see “BPXYMNTE — Map response and element structure of w\_getmntent” on page 1104.

If BPXYMNTE is assembled with MNTE2=YES, fields MNTEHID and MNTEHLEN must be initialized. AMODE 31 callers use “BPX1GMN (w\_getmntent) example” on page 1246.

LA	R14,MNTEH	R14->MNTEH and MNTE	
L	R15,MNTEL	R15 = Length of MNTEH and MNTE	
XR	R0,R0	Dummy 2nd operand	
XR	R1,R1	Pad=null, length=0	
MVCL	R14,R0	Null out MNTEH and MNTE	
MVC	MNTEHID,=CL4'MNT2'	Version indicator	
MVC	MNTEHLEN,=A(MNTE#LENGTH)	Length of MNTE	
CALL	BPX4GMN,	Get mount entries	+
	(MNTEL,	Input: Length BPXYMNTE + MNTEH	+
	MNTEH,	Header in BPXYMNTH	+
	RETVAL,	Return value: -1 or mount entries	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4GNI (getnameinfo) example

The following code resolves a socket address into a host name and a service name. For the callable service, see “getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address” on page 291. AMODE 31 callers use “BPX1GNI (getnameinfo) example” on page 1246.

SPACE ,			
CALL	BPX4GNI,	Get name info	+
	(SOCKADDR,	Input: Socket address	+
	SOCKADDR_LENGTH,	Input: Length of socket address	+
	SERVICE_BUFFER,	I/O: Buffer for service name	+
	SERVICE_BUFFER_LENGTH,	I/O: Length of service buffer	+
	HOST_BUFFER,	I/O: Buffer for host name	+
	HOST_BUFFER_LENGTH,	I/O: Length of host buffer	+
	FLAGS,	Input: Flags	+
	RETVAL,	Return code	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4GPG (getpgrp) example

The following code gets the process group ID of the caller. For the callable service, see “getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID” on page 297. AMODE 31 callers use “BPX1GPG (getpgrp) example” on page 1247.

CALL	BPX4GPG,	Get the process group ID	+
	(RETVAL),	Return value: group ID	+
	MF=(E,PLIST)	-----	

## BPX4GNM (getpeername or getsockname) example

The following code gets the peer name, and then requests the socket name. SOCKDESC was returned by a previous call to BPX4SOC. For the callable service, see “getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer” on page 319. For the data structure,

see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1GNM (getpeername or getsockname) example” on page 1247.

```

SPACE ,
CALL BPX4GNM,           Get peername           +
      (SOCKDESC,        Input: Socket Descriptor +
      SOCK#GNM0PTGETPEERNAME, Input: Indicate getpeername +
      SOCK#LEN+SOCK_SUN#LEN, Input: Length - Sockaddr +
      SOCKADDR,         Input: Sockaddr structure +
      RETVAL,           Return value: 0 or -1     +
      RETCODE,          Return code              +
      RSNCODE),         Reason code              +
      MF=(E,PLIST)      -----
SPACE ,
CALL BPX4GNM,           Get sockname           +
      (SOCKDESC,        Input: Socket Descriptor +
      SOCK#GNM0PTGETSOCKNAME, Input: Indicate getpeername +
      SOCK#LEN+SOCK_SUN#LEN, Input: Length - Sockaddr +
      SOCKADDR,         Input: Sockaddr structure +
      RETVAL,           Return value: 0 or -1     +
      RETCODE,          Return code              +
      RSNCODE),         Reason code              +
      MF=(E,PLIST)      -----

```

## BPX4GPE (getpwent) example

The following code accesses the user database starting with the next available entry and continuing until end of file on the database. It returns a structure identifying information about each user entry in the database. For the callable service, see “getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database” on page 302. For the data structure, see “BPXYGIDN — Map data returned for getpwnam and getpwuid” on page 1079. AMODE 31 callers use “BPX1GPE (getpwent) example” on page 1247.

```

GPELOOP DS 0H
CALL BPX4GPE,           Access the user database +
      (RETVAL,          Return value: 0 or ->BPXYGIDN +
      RETCODE,          Return code              +
      RSNCODE),         Reason code              +
      MF=(E,PLIST)      -----
      ICM R8,B'1111',RETVAL
      BZ CHKGPERR        Error or end of file
      USING GIDN,R8
*      access the user structure
      DROP R8
      B GPELOOP          Check next user entry
CHKGPERR DS 0H
      ICM R8,B'1111',RETCODE
      BZ GPPEOF          End of file
*      handle error as needed
GPPEOF DS 0H

```

## BPX4GPI (getpid) example

The following code gets the process ID of the caller. For the callable service, see “getpid (BPX1GPI, BPX4GPI) — Get the process ID” on page 298. AMODE 31 callers use “BPX1GPI (getpid) example” on page 1248.

```

CALL BPX4GPI,           Get the process ID       +
      (RETVAL),         Returns value, Process ID +
      MF=(E,PLIST)      -----

```

## BPX4GPN (getpwnam) example

The following code accesses the group database by the user ID of the caller and returns a structure identifying the groups by name. For the callable service, see “getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name” on page 305. For the data structure, see “BPXYGIDN — Map data

## BPX4GPP (getppid) example

returned for getpwnam and getpwuid” on page 1079. AMODE 31 callers use “BPX1GPN (getpwnam) example ” on page 1248.

```
MVC  USERNLEN,=F'8'
MVC  USERNAME(8),=CL8'Pebbles'
SPACE ,
CALL  BPX4GPN,          Access the user database      +
      (USERNLEN,        Input: Length of user name    +
      USERNAME,         Input: Name of user           +
      RETVAL,           Return value 0 or ->BPXYGIDN    +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      MF=(E,PLIST)      -----
```

## BPX4GPP (getppid) example

The following code gets the process ID of the caller's parent. For the callable service, see “getppid (BPX1GPP, BPX4GPP) — Get the parent process ID ” on page 299. AMODE 31 callers use “BPX1GPP (getppid) example” on page 1248.

```
CALL  BPX4GPP,          Get PID of the parent process  +
      (RETVAL),         Returns value, parent's process ID+
      MF=(E,PLIST)      -----
```

## BPX4GPT (grantpt) example

The following code grants access to the subsidiary pseudoterminal device that is identified by the file descriptor. For the callable service, see “grantpt (BPX1GPT, BPX4GPT) — Grant access to the subsidiary pseudoterminal” on page 336. AMODE 31 callers use “BPX1GPT (grantpt) example” on page 1249.

```
CALL  BPX4GPT,          Grant access to subsidiary pty  +
      (MASTER_FD,      Input: File descriptor          +
      RETVAL,           Return value: 0 or -1           +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      MF=(E,PLIST)      -----
```

## BPX4GPU (getpwuid) example

The following code accesses the group database by the user name of the caller and returns a structure identifying the groups by name. The code sets the user ID value to 1. For the callable service, see “getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID ” on page 308. For the data structure, see “BPXYGIDN — Map data returned for getpwnam and getpwuid” on page 1079. AMODE 31 callers use “BPX1GPU (getpwuid) example” on page 1249.

```
MVC  USERID,..         Value of user ID
SPACE ,
CALL  BPX4GPU,          Access database by user ID      +
      (USERID,          Input: User ID                 +
      RETVAL,           Return value 0 or ->BPXYGIDN    +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      MF=(E,PLIST)      -----
```

## BPX4GPY (getpriority) example

The following code gets the CPU priority based on the input which and who values. The which value used is PRIO\_PROCESS, which indicates to get the priority by process ID. The who value used is 7, indicating to get the priority for process ID 7. For the callable service, see “getpriority (BPX1GPY, BPX4GPY) — Get the

scheduling priority of a process ” on page 300. AMODE 31 callers use [“BPX1GPY \(getpriority\) example”](#) on page 1249.

```

MVC PROCID,=XL4'00000007' Process ID to get priority for
SPACE ,
CALL BPX4GPY,          Get priority value          +
      (=A(PRIO_PROCESS), Input: Get by Process ID    +
      PROCID,          Input: PID to get priority for +
      RETVAL,          Return value: Priority of process +
      RETCODE,         Return code                  +
      RSNCODE),        Reason code                  +
      MF=(E,PLIST)     -----
L   R15,RETVAL          Load return value
C   R15,=F'-1'         Test for -1 return
BE  PSEUDO             Branch on error

```

## BPX4GRD (getrandom) example

The following code creates a **getrandom** instance. For the callable service, use [“getrandom \(BPX1GRD, BPX4GRD\) — Obtain a series of random bytes”](#) on page 310. AMODE 31 callers use the [“BPX1GRD \(getrandom\) example”](#) on page 1250.

```

MVC FLAGS,=F'2' Flags
MVC BUFLN,=F'4' Address length
LA R15,BUFFERA Space for BUFLN words
ST R15,BUFADR Buffer address
CALL BPX4GRD,          Get random +
      (FLAGS,          Input: FLAGS +
      BUFLN,           Input: Buffer Length +
      BUFADR,          Input: Buffer Address +
      RETVAL,          Return value: random value +
      RETCODE,         Return code +
      RSNCODE),        Reason code +
      VL,MF=(E,PLIST) -----
L   R15,RETVAL          Load return value
C   R15,=F'-1'         Test for -1 return

```

## BPX4GRL (getrlimit) example

The following code fills in the rlimit structure for the calling process based on the input resource value. The resource value is set to RLIMIT\_CPU. For the callable service, see [“getrlimit \(BPX1GRL, BPX4GRL\) — Get resource limits ”](#) on page 313. For the data structure, see [“BPXYRLIM — Map the rlimit, rusage, and timeval structures”](#) on page 1142. AMODE 31 callers use [“BPX1GRL \(getrlimit\) example”](#) on page 1250.

```

MVC RESOURCE,=A(RLIMIT_CPU) Value of resource
SPACE ,
CALL BPX4GRL,          Get resource limits          +
      (RESOURCE,       Input: resource              +
      RLIMIT,          Structure, mapped by BPXYRLIM +
      RETVAL,          Return value: 0 or -1         +
      RETCODE,         Return code                  +
      RSNCODE),        Reason code                  +
      MF=(E,PLIST)     -----
L   R15,RETVAL          Load return value
C   R15,=F'-1'         Test for -1 return
BE  PSEUDO             Branch on error

```

## BPX4GRU (getrusage) example

The following code fills in the rusage structure based on the input who value. The who value is set to RUSAGE\_SELF. For the callable service, see [“getrusage \(BPX1GRU, BPX4GRU\) — Get resource usage ”](#) on

## BPX4GTH (getitimer) example

page 315. For the data structure, see “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142. AMODE 31 callers use “BPX1GRU (getrusage) example” on page 1250.

```
MVC  WHO,=A(RUSAGE_SELF)  Value of who
SPACE ,
CALL BPX4GRU,              Get resource usage          +
    (WHO,                  Input: who                  +
     RUSAGE,               Structure, mapped by BPXYRLIM +
     RETVAL,              Return value: 0 or -1          +
     RETCODE,             Return code                  +
     RSNCODE),            Reason code                  +
    MF=(E,PLIST)          -----
L    R15,RETVAL            Load return value
C    R15,=F'-1'           Test for -1 return
BE   PSEUDO               Branch on error
```

## BPX4GTH (\_\_getthent) example

The following code retrieves information on the first process accessible to the caller. For the callable service, see “\_\_getthent (BPX1GTH, BPX4GTH) — Get thread data” on page 325. For the data structure, see “BPXYPGTH — Map the \_\_getthent input/output structure” on page 1119. AMODE 31 callers use “BPX1GTH (\_\_getthent) example” on page 1251.

```
LA    R5,BUFFERB          R5 -> Input parameters
STG   R5,BUFB             ->input buffer
USING PGTHA,R5            R5 base for PGTHA
XC    PGTHA(PGTHA#LEN),PGTHA Null Input area
MVI   PGTHAFLAG1,PGTHAPROCESS+PGTHACOMMAND+PGTHATHREAD
MVI   PGTHAACCESSSTHID,PGTH#LAST Last thread
LA    R15,BUFFERA         PgthB, Output buffer
STG   R15,BUFA            ->output buffer
DROP  R5
SPACE ,
CALL  BPX4GTH,            __getthent                  +
    (=A(PGTHA#LEN),      Input: length input parms BPXYPGTH+
    BUFA,                Input: ->input parms        BPXYPGTH+
    =A(1024),            Input: length output area   BPXYPGTH+
    BUFB,                Input: ->output area        BPXYPGTH+
    RETVAL,              Return value: 0, -1          +
    RETCODE,             Return code                  +
    RSNCODE),            Reason code                  +
    MF=(E,PLIST)          -----
```

## BPX4GTR (getitimer) example

The following code returns the time remaining an alarm, or ITIMER\_REAL as set by setitimer. For the callable service, see “getitimer (BPX1GTR, BPX4GTR) — Get the value of the interval timer” on page 287. For the data structure, see “BPXYITIM — Map getitimer, setitimer structure ” on page 1101. AMODE 31 callers use “BPX1GTR (getitimer) example” on page 1251.

```
LA    R15,ITIM            Output mapping structure
STG   R15,ITIMA           ->structure
CALL  BPX4GTR,            Get process data            +
    (=A(ITIMER_REAL),    Input: Relative process token +
    ITIMA,               Out: ->Buffer, mapped by BPXYITIM +
    RETVAL,              Return value: -1, 0            +
    RETCODE,             Return code                    +
    RSNCODE),            Reason code                    +
    MF=(E,PLIST)          -----
```

## BPX4GUG (getgroupsbyname) example

The following code returns the number of supplementary group IDs, up to 9, for user Pebbles. For the callable service, see “getgroupsbyname (BPX1GUG, BPX4GUG) — Get a list of supplementary group IDs

by user name” on page 276. AMODE 31 callers use “BPX1GUG (getgroupsbyname) example” on page 1251.

```

MVC  USERLEN,=F'7'
MVC  USERNAME(07),=CL07'Pebbles'
MVC  BUFLINA,=F'9'
LA   R15,BUFFERA
STG  R15,BUFA
SPACE
CALL  BPX4GUG,          Get list of groups by user name  +
      (USERLEN,         Input: User name length        +
      USERNAME,         Input: User name                +
      BUFLINA,          Input: Group ID list size        +
      BUFA,             Group ID list address           +
      RETVAL,           Return value: -1, or # of grp IDs +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4GUI (getuid) example

The following code gets the invoker's real user ID. For the callable service, see “getuid (BPX1GUI, BPX4GUI) — Get the real user ID ” on page 329. AMODE 31 callers use “BPX1GUI (getuid) example” on page 1252.

```

CALL  BPX4GUI,          Get the real user ID            +
      (RETVAL),         Return value: real user ID      +
      MF=(E,PLIST)      -----

```

## BPX4GWD (getwd) example

The following code gets the working directory for the caller. For the callable service, see “getwd (BPX1GWD, BPX4GWD) — Get the path name of the working directory” on page 330. AMODE 31 callers use “BPX1GWD (getwd) example” on page 1252.

```

MVC  BUFLINA,=F'1024'   Max directory name return area
SPACE
CALL  BPX4GWD,          Get working directory name      +
      (BUFLINA,         Input: Length directory work area +
      BUFFERA,          Buffer                          +
      RETVAL,           Return value: length or -1       +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4HST (gethostid or gethostname) example

The following code requests the host id and the host name for an AF\_INET domain. For the callable service, see “gethostid or gethostname (BPX1HST, BPX4HST) — Get ID or name information about a socket host” on page 285. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1HST (gethostid or gethostname) example” on page 1252.

```

XC   BUFLINA,BUFLINA
CALL BPX4HST,          Request host id                  +
      (=A(AF_INET),    Input: Domain - AF_INET          +
      BUFLINA,         Input: Length - No buffer - get id+
      BUFFERA,         Output: (not used with Length=0) +
      RETVAL,          Return value: 0 or -1            +
      RETCODE,         Return code                    +
      RSNCODE),        Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4IAW (inotify\_add\_watch)

MVC	BUFLINA,=A(L'BUFFERA)		
CALL	BPX4HST,	Request host name	+
	(=A(AF_INET),	Input: Domain - AF_INET	+
	BUFLINA,	Input: Length - for output name	+
	BUFFERA,	Output: Buffer for host name	+
	RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4IAW (inotify\_add\_watch) example

The following code creates an **inotify\_add\_watch** instance. For the callable service, use “[inotify\\_add\\_watch \(BPX1IAW, BPX4IAW\) — Add or modify a watch to an initialized inotify instance](#)” on page 338. AMODE 31 callers use the “[BPX1IAW \(inotify\\_add\\_watch\) example](#)” on page 1252.

```
XC  aOptions(4),aOptions
MVI  iOptions,IN_ALL_EVENTS
CALL BPX4IAW(      Add-Watch a file/directory
  inotifyFD,      Inotify Instance FD from BPXUSINI
  aPathNnameLen,  Length of the input pathname
  iPathName,      PathName: i.e '/inotify/dir1'.
  aOptions,       Mask: IN_ALL_EVENTS: BPXYINFY
  RetVal,         Return Value Inotify Watch Descriptor or -1
  RetCode,        Return Code
  RsnCode);       Reason Code
L    @05,RetVal   Loading Return Value
CHI  @05,-1       Testing RetVal
BE   ERROR        Branch if negative (-1=failure)
ST   @05, INOTIFYWD Save Inotify Watch Descriptor
```

## BPX4INI (inotify\_init) example

The following code creates an **inotify\_init** instance. For the callable service, use “[inotify\\_init \(BPX1INI, BPX4INI\) — Initialize an inotify instance](#)” on page 341. AMODE 31 callers use the “[BPX1INI \(inotify\\_init\) example](#)” on page 1253.

```
XC  iOptions(4)iOptions
MVI  iOptions,IN_NONBLOCK
CALL BPX4INI(      Create an inotify instance FD      +
  iOptions,        Inotify Instance Option: BPXYIFY  +
  RETVAL,          Return value Inotify FD or -1     +
  RETCODE,         Return code                      +
  RSNCODE),        Reason code                      +
L    @05,RetVal    Loading return value
CHI  @05,-1        Testing RetVal
BE   ERROR         Branch if negative (-1 = failure)
ST   @05,INOTIFYFD Save Inotify File Descriptor
```

## BPX4IRM (inotify\_rm\_watch) example

The following code creates an **inotify\_rm\_watch** instance. For the callable service, use “[inotify\\_rm\\_watch \(BPX1IRW, BPX4IRW\) — Remove existing watches from inotify instances](#)” on page 344. AMODE 31 callers use the “[BPX1IRM \(inotify\\_rm\\_watch\) example](#)” on page 1254.

```
CALL BPX4IRW(      Remove-Watch of a file/directory
  inotifyFD,        Inotify Instance FD from BPXUSINI
  inotifyWD,        Inotify Watch-Descriptor from BPXUSIAW
  RetVal,          Return Value 0 or -1
  RetCode,         Return Code
  RsnCode);       Reason Code
L    @05,RetVal    Loading Return Value
CHI  @05,-1        Testing RetVal
BE   ERROR         Branch if negative (-1=failure)
```



## BPX4IOC (w\_iocctl) example

The following code conveys a command to the standard output device. To run properly this example needs a command defined by the user for the COMMAND parameter. This command must be understood by the device driver providing support for the output device. For the callable service, see “w\_iocctl (BPX1IOC, BPX4IOC) – Control I/O ” on page 1003. AMODE 31 callers use “BPX1IOC (w\_iocctl) example” on page 1253.

MVC	BUFLINA,=F'1024'		
MVC	COMMAND,=F'123'	User defined command	
SPACE	,		
CALL	BPX4IOC,	I/O Control	+
	(=A(STDOUT_FILENO),	Input: File descriptor	+
	COMMAND,	Input: Command	+
	BUFLINA,	Input: Argument length	+
	BUFFERA,	Argument buffer name	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4IPT (mvsiptaffinity) example

The following code executes the assembler routine EXITRTN on the IPT of the requesting thread, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see “mvsiptaffinity (BPX1IPT, BPX4IPT) – Run a program on the IPT thread ” on page 476. AMODE 31 callers use “BPX1IPT (mvsiptaffinity) example” on page 1253.

*	MVC	EXITRTNA,=AD(EXITRTN)	->Routine address	
	MVC	EXITPLA,=AD(EXITPARM)	->Input parameter list	
	SPACE	,		
	CALL	BPX4IPT,		+
		(EXITRTNA,	Input: Routine address	+
		EXITPLA,	Input: Parm list address or 0	+
		RETVAL,	Return value: -1 or not return	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		MF=(E,PLIST)	-----	

## BPX4ITY (isatty) example

The following code determines if the standard output device is a terminal. For the callable service, see “isatty (BPX1ITY) (POSIX Version) – Determine whether a file descriptor represents a terminal” on page 357. AMODE 31 callers use “BPX2ITY (isatty) example ” on page 1254.

	CALL	BPX4ITY,	Determine if device is a TTY	+
		(=A(STDOUT_FILENO),	Input: File descriptor	+
		RETVAL,	Return value: 0 isn't, 1 is, -1	+
		RETCODE,	Return code: describes why VAL=-1	+
		RSNCODE),	Reason code: qualifier on RETCODE	+
		MF=(E,PLIST)	-----	
	ICM	R15,B'1111',RETVAL	Test RETVAL	
	BZ	PSEUDO	RETVAL=0 means device not terminal	

## BPX4KIL (kill) example

The following code sends a signal (SIGUSR1) to all processes for which access is allowed in the invoker's process group. For the callable service, see “kill (BPX1KIL, BPX4KIL) – Send a signal to a process ” on page 360. For the data structure, see “BPXYSIGH – Signal constants” on page 1148. AMODE 31 callers use “BPX1KIL (kill) example” on page 1254.

MVC	PROCID,=A(0)	Invoker's process group
-----	--------------	-------------------------

## BPX4LCO (lchown) example

CALL	BPX4KIL,	Send a signal to a process	+
	(PROCID,	Input: Process ID	+
	=A(SIGUSR1#),	Input: Signal	BPXYSIGH +
	=A(0),	Input: Signal options	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4LCO (lchown) example

The following code changes the owner of symbolic link /somedir/somesymlink.c from the current owner to that specified by USERID and GROUPID. For the callable service, see [“lchown \(BPX1LCO, BPX4LCO\) — Change the owner or group of a file, directory, or symbolic link”](#) on page 380. AMODE 31 callers use [“BPX1LCO \(lchown\) example”](#) on page 1255.

MVC	BUFFERA(22),=CL22'/somedir/somesymlink.c'		
MVC	BUFLINA,=F'22'		
MVC	USERID,..	New owner UID from stat	
MVC	GROUPID,..	New owner GID from stat	
	SPACE,		
CALL	BPX4LCO,	Change owner and group of a file	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	USERID,	Input: New owner UID	+
	GROUPID,	Input: New owner GID	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4LCR (lchattr) example

The following code changes the attributes of symbolic link /somedir/somesymlink.c. The security label is set and the file change time is set. For the callable service, see [“lchattr \(BPX1LCR, BPX4LCR\) — Change the attributes of a file or directory or symbolic link”](#) on page 371. For the data structures, see [“BPXYATT — Map file attributes for chattr and fchattr”](#) on page 1057. AMODE 31 callers use [“BPX1LCR \(lchattr\) example”](#) on page 1255.

MVC	BUFFERA(22),=CL22'/somedir/somesymlink.c'		
MVC	BUFLINA,=F'22'		
XC	ATT,ATT	Clear ATT	
MVC	ATTID,=CL4'ATT '	Eye Catcher	
MVC	ATTVERSION,=AL2(ATT#VER)	version	
MVC	ATTSECLABEL,=CL08'SYSMULTI'		
OI	ATTSETFLAGS3,ATTSECLABELCHG		+
	Flag Seclabel update		
OI	ATTSETFLAGS2,ATTCTIMETOD		+
	Set change time to current time		
	SPACE,		
CALL	BPX4LCR,	Change file attributes	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	=A(ATT#LENGTH),	Input: BPXYATT length	+
	ATT,	Input/output: BPXYATT	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4LDX (loadhfs extended) example

The following is an example specifying the Lod\_Directed option. For an example of BPX1LDX/BPX4LDX without the Lod\_Directed option flag specified, see [“BPX1LOD \(loadhfs\) example”](#) on page 1257, substituting BPX1LDX/BPX4LDX for BPX1LOD/BPX4LOD. The program ictasma located at ict/bin is loaded into storage and then branched to. Then the CSVDYLPA service is called to provide serviceability information to the system. The loaded module can then be branched to. When the load module is no

longer needed, the serviceability information should be deleted and the module's storage released. For the callable service, see [“loadhfs extended \(BPX1LDX, BPX4LDX\) — Direct the loading of an executable into storage”](#) on page 399. AMODE 31 callers use [“BPX1LDX \(loadhfs extended\) example”](#) on page 1255.

```

MVC  BUFLINA,=F'16'
MVC  BUFFERA(16),=C'/ict/bin/ictasma'
MVC  OPTIONS,=AL4(LOD_DIRECTED)  Directed loadhfs to common
OI   OPTIONS+3,X'F1'             Subpool 241
MVC  LIBPHTLN,=A(0)
SPACE ,
CALL  BPX4LDX,  Load program
          (BUFLINA, Input: Pathname length          +
          BUFFERA,  Input: Pathname                  +
          OPTIONS,  Input: Options                    +
          LIBPHTLN, Input: Library Path Length        +
          LIBPATH,  Input: Library Path                +
          RTNPARM@, Output: directed load ret parm structure +
          RETVAL,   Return value: -1 or 0              +
          RETCODE,  Return code                        +
          RSNCODE), Reason code                      +
          MF=(E,PLIST) -----
SPACE ,
L     R15,RETVAL  Load return value
C     R15,=F'-1'  Test for -1 return
BE    PSEUDO     Branch on error
SPACE ,
LG    R5,RTNPARM@
MVC   LOCALPARMS(24),0(R5)  Local copy of returned parameters
*
*       Provide serviceability information to system
*
LA    R4,LOCALPARMS
USING DIRECTEDLOADRETURNEDPARMS,R4
LGHI  R7,-2
LG    R5,DIRECTEDLOADMODULEENTRYPT64
NGR   R5,R7      Clear entry point amode 64 flag
STG   R5,DIRECTEDLOADMODULEENTRYPT64
LG    R6,DIRECTEDLOADMODULESTART64
LG    R7,DIRECTEDLOADMODULELENGTH64
XC    LPMEA(LPMEA_LEN),LPMEA
ST    R5,LPMEAENTRYPOINTADDR
ST    R6,LPMEALOADPOINTADDR
ST    R7,LPMEAMODLEN
MVC   LPMEANAME,=C'ICTASMA '
CSVDYLP REQUEST=ADD,
          BYADDR=YES,
          MODINFOTYPE=MEMBERLIST,
          MODINFO=LPMEA,
          NUMMOD=1,
          REQUESTOR=REQID,
          RETCODE=RETCODE,
          RSNCODE=RSNCODE,
          MF=(E,DYLPAL)  Provide serviceability information
L     R15,RETCODE  Load return code
LTR   R15,R15
BNZ   PSEUDO
MVC   LOCALDELTOKEN(8),LPMEADELETETOKEN
SPACE ,
.
.
.
*
*       Call directed loadhfs target module
*
LG    R15,DIRECTEDLOADMODULEENTRYPT64
BALR  R14,R15  Branch to loaded program
SPACE ,
.
.
.
*
*       When done with directed load hfs module
*       remove serviceability information and
*       release module storage
*
XC    LPMED(LPMED_LEN),LPMED
MVC   LPMEDNAME,=C'ICTASMA '
MVC   LPMEDDELETETOKEN(8),LOCALDELTOKEN
CSVDYLP REQUEST=DELETE,

```

## BPX4LNA (linkat) example

```

                                TYPE=BYTOKEN,
                                MODINFO=LPMED,
                                NUMMOD=1,
                                RETCODE=RETCODE,
                                RSNCODE=RSNCODE,
                                MF=(E,DYLPAL) Remove serviceability information
L      R15,RETCODE             Load return code
LTR    R15,R15
BNZ    PSEUDO
SPACE ,
MODESET MODE=SUP
L      R7,DIRECTEDLOADMODULELENGTH
STORAGE RELEASE,
                                LENGTH=(R7),
                                ADDR=DIRECTEDLOADMODULESTART,
                                SP=241 Free module
MODESET MODE=PROB
DROP   R4
```

## BPX4LNA (linkat) example

The following code creates a hard link to an existing file `lnk03oldfile01.txt`. For the callable service, see [“linkat \(BPX1LNA, BPX4LNA\) — Make a link using at directory relative to a directory file descriptor”](#) on page 387. For the data structure, see [“BPXYFCTL — Command values and flags for fcntl”](#) on page 1076. AMODE 31 callers use the [“BPX1LNA \(linkat\) example”](#) on page 1257.

```

CALL   BPX4LNA,                Create a hard link to a file    x
      (=F'18',                  Length of existing file name  x
      =C'lnk03oldfile01.txt',   Name of exiting file         x
      =F'19',                  Length of new path name        x
      =C'lnk03_newfile02.txt',  Name of new path name        x
      Olddirfd,                Directory for old file         x
      newdirfd,                Directory for new file         x
      =A(At_SymLink_Follow),   Flags                        x
      retVal,                  x
      my_retCode,              x
      rsnCode),               x
      VL,MF=(E,PLIST)
PLIST  DS    10A
      BPXYFCTL
```

## BPX4LOD (loadhfs) example

The program `ictasma` located at `ict/bin` is loaded into storage and then branched to. For the callable service, see [“loadhfs \(BPX1LOD, BPX4LOD\) — Load a program into storage by path name”](#) on page 393. AMODE 31 callers use [“BPX1LOD \(loadhfs\) example”](#) on page 1257.

```

MVC    BUFLNA,=F'16'
MVC    BUFFERA(16),=C'/ict/bin/ictasma'
MVC    OPTIONS,=A(0)
MVC    LIBPHTLN,=A(0)
SPACE ,
CALL   BPX4LOD,                Load program
      (BUFLNA,                  Input: Pathname length
      BUFFERA,                  Input: Pathname
      OPTIONS,                  Input: Options
      LIBPHTLN,                 Input: Library Path Length
      LIBPATH,                  Input: Library Path
      ENTRYPT,                  Output: Entry Point
      RETVAL,                   Return value: -1 or 0
      RETCODE,                  Return code
      RSNCODE),                 Reason code
      MF=(E,PLIST)
SPACE ,
L      R15,RETVAL
C      R15,=F'-1'              Test for -1 return
BE     PSEUDO                   Branch on error
SPACE ,
LG     R15,ENTRYPT
BALR   R14,R15                 Branch to loaded program
```

## BPX4LNK (link) example

The following code creates a new way for `usr/dataproc.next.t` to link to an existing file, `usr/user05/yearrecs.t`. For the callable service, see “[link \(BPX1LNK, BPX4LNK\) — Create a link to a file](#)” on page 383. AMODE 31 callers use “[BPX1LNK \(link\) example](#)” on page 1258.

```

MVC  BUFLINA,=F'21'
MVC  BUFFERA(21),=CL21'usr/user05/yearrecs.t'
MVC  BUFLINB,=F'19'
MVC  BUFFERB(19),=CL19'usr/dataproc.next.t'
SPACE ,
CALL  BPX4LNK,          Create a link to a file          +
      (BUFLINA,          Input: Name length: existing    +
      BUFFERA,           Input: Name of existing file     +
      BUFLINB,           Input: Name length: link         +
      BUFFERB,           Input: Name of link to file      +
      RETVAL,            Return value: 0 or -1            +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                     +
      MF=(E,PLIST)      -----

```

## BPX4LSK (lseek) example

The following code changes the file (FILEDESC) offset to 80 bytes past the current offset. For the callable service, see “[lseek \(BPX1LSK, BPX4LSK\) — Change a file's offset](#)” on page 406. For the data structure, see “[BPXYSEEK — Constants for lseek](#)” on page 1144. AMODE 31 callers use “[BPX1LSK \(lseek\) example](#)” on page 1258.

```

MVC  FILEDESC,...      File descriptor from open
MVC  OFFSET(08),=FL8'80' Forward 80 Bytes
MVC  REFPT,=A(SEEK_CUR) Current offset of the file
SPACE ,
CALL  BPX4LSK,          Change a file's offset          +
      (FILEDESC,         File descriptor                +
      OFFSET,            I/O: Offset in file            +
      REFPT,             Input: Reference point, BPXYSEEK +
      RETVAL,            Return value: 0 or -1          +
      RETCODE,           Return code                    +
      RSNCODE),          Reason code                   +
      MF=(E,PLIST)      -----

```

## BPX4LSN (listen) example

The following code issues a listen on a socket that was previously created and that had a bind done for it. SOCKDESC was returned from the call to BPX4SOC. Set the backlog count to 5. For the callable service, see “[listen \(BPX1LSN, BPX4LSN\) — Prepare a server socket to queue incoming connection requests from clients](#)” on page 391. For the data structure, see “[BPXYSOCK — Map SOCKADDR structure and constants](#)” on page 1151. AMODE 31 callers use “[BPX1LSN \(listen\) example](#)” on page 1258.

```

CALL  BPX4LSN,          Listen on a socket              +
      (SOCKDESC,         Input: Socket Descriptor        +
      =A(5),             Input: Backlog count of 5        +
      RETVAL,            Return value: 0 or -1            +
      RETCODE,           Return code                      +
      RSNCODE),          Reason code                     +
      MF=(E,PLIST)      -----

```

## BPX4LST (lstat) example

The following code obtains the file status for the file described by the symbolic name `labrec/sym`. For the callable service, see “[lstat \(BPX1LST, BPX4LST\) — Get status information about a file or symbolic link](#)” on page 383.

## BPX4MAT (shmat) example

by path name” on page 409. For the data structure, see “BPXYSTAT — Map the response structure for stat” on page 1162. AMODE 31 callers use “BPX1LST (lstat) example” on page 1258.

```
* symbolic name established using symlink (BPX4SYM) system call
MVC  BUFFERA(10),=CL10'labrec/sym'
MVC  BUFLINA,=F'10'
SPACE ,
CALL  BPX4LST,          Get file status          +
      (BUFLINA,         Input: Pathname length    +
      BUFFERA,          Input: Pathname          +
      STATL,            Input: Length of buffer needed +
      STAT,             Buffer, mapped by BPXYSTAT  +
      RETVAL,           Return value: 0 or -1      +
      RETCODE,          Return code               +
      RSNCODE),         Reason code               +
      MF=(E,PLIST)      -----
```

## BPX4MAT (shmat) example

The following code attaches a shared memory segment. For the callable service, see “shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment” on page 804. For the data structure, see “BPXYSHM — Map interprocess communication shared memory segments” on page 1147. AMODE 31 callers use “BPX1MAT (shmat) example” on page 1259.

```
CALL  BPX4MAT,          Shared memory segment control +
      (SHM_ID,          Input: Shared memory segment ID +
      SEGADDR,          Input: ST loc for seg address  +
      =A(0),            Input: Flags                  BPXYSHM +
      ATTADDR,          Output: memory segment address +
      RETVAL,           Return value: 0, -1            +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      MF=(E,PLIST)      -----
```

## BPX4MCT (shmctl) example

The following code retrieves the size of the shared memory segment. For the callable service, see “shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations” on page 809. For the data structure, see “BPXYSHM—Map interprocess communication shared memory segments” on page 1147. AMODE 31 callers use “BPX1MCT (shmctl) example” on page 1259.

```
LA    R15,BUFFERA
STG   R15,BUFA
SPACE ,
CALL  BPX4MCT,          Shared memory segment control +
      (SHM_ID,          Input: Shared memory segment ID +
      =A(IPC_STAT),     Input: Command                BPXYIPC +
      BUFA,             Input: ->SHMID_DS or 0        BPXYSHM +
      RETVAL,           Return value: 0, -1 or value    +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      MF=(E,PLIST)      -----
```

## BPX4MDA (mkdirat) example

The following code creates a new and empty directory pathname of /usr/newprots/ with user read-execute, group write, and other read-execute permissions. For the callable service, see “mkdirat (BPX1MDA, BPX4MDA) — Create a directory relative to a directory file descriptor” on page 423. For the data structure, see “BPXYFTYP — File type definitions” on page 1078 and “BPXYMODE — Map the mode constants” on page 1106. AMODE 31 callers use “BPX1MDA (mkdirat) example” on page 1259.

```
MVC  BUFLINA,=F'7'
MVC  BUFFERA(7),=CL7'testdir'
XC   S_MODE,S_MODE
```

```

MVI S_MODE2,S_IRUSR
MVI S_MODE3,S_IRWXU2+S_IRGRP+S_IROTH
CALL BPX4MDA      mkdirat
      (BUFLNA,      Input: Path Length
      BUFFERA,      Input: Path Name Address
      S_MODE,        Input: Mode, mapped by BPXYMODE
      FILEDESC,      Input: Directory file descriptor
      RETVAL,        Return value: 0 or -1
      RETCODE,       Return code
      RSNCODE),      Reason code
      VL,MF=(E,PLIST) -----
L    R15,RETVAL      Load Return Value
C    R15,-F'1'       Test for -1 return value
BE   PSEUDO0E        Branch on Error path
LR   R4,R15          Save return value

```

## BPX4MDT (shmdt) example

The following code detaches a shared memory segment. For the callable service, see [“shmdt \(BPX1MDT, BPX4MDT\) — Detach a shared memory segment”](#) on page 812. For the data structure, see [“BPXYSHM — Map interprocess communication shared memory segments”](#) on page 1147. AMODE 31 callers use [“BPX1MDT \(shmdt\) example”](#) on page 1260.

```

CALL BPX4MDT,      Shared memory segment detach      +
      (SEGADDR,    Input: Shared memory segment addr +
      RETVAL,      Return value: 0, -1 or value      +
      RETCODE,     Return code                      +
      RSNCODE),    Reason code                      +
      MF=(E,PLIST) -----

```

## BPX4MGT (shmget) example

The following code creates a private shared memory segment of 500 bytes. For the callable service, see [“shmget \(BPX1MGT, BPX4MGT\) — Create/find a shared memory segment”](#) on page 827. For the data structure, see [“BPXYSEM — Map interprocess communication semaphores”](#) on page 1146. AMODE 31 callers use [“BPX1MGT \(shmget\) example”](#) on page 1260.

```

MVC KEY(4),=A(IPC_PRIVATE) Local to this family
MVI S_TYPE,IPC_CREAT+IPC_EXCL Must not already exist
MVI S_MODE1,0                Not used
MVI S_MODE2,S_IRUSR          All read and write permissions
MVI S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL BPX4MGT,                Create a set of semaphores      +
      (KEY,              Input: Shared memory segment KEY    +
      =AD(500),          Input: Segment size                 +
      S_MODE,            Input: Creation flags               BPXYIPC +
      RETVAL,            Return value: -1 or MessageQueue ID +
      RETCODE,           Return code                         +
      RSNCODE),          Reason code                         +
      MF=(E,PLIST)      -----
SPACE ,
ICM R15,B'1111',RETVAL      Test return value
BNP PSEUDO                  Branch on shmget failure
ST  R15,SHM_ID              Store SHM_ID associated with key

```

## BPX4MKD (mkdir) example

The following code creates a new and empty directory pathname of /usr/newprots/ with user read-execute, group write, other read-execute permissions. For the callable service, see [“mkdir \(BPX1MKD, BPX4MKD\) — Make a directory”](#) on page 420. For the data structure, see [“BPXYFTYP — File type definitions”](#) on page 1078 and [“BPXYMODE — Map the mode constants”](#) on page 1106. AMODE 31 callers use [“BPX1MKD \(mkdir\) example”](#) on page 1260.

```

MVC BUFFERA(14),=CL14' /usr/newprots/'
MVC BUFLNA,=F'14'

```

## BPX4MKN (mknod) example

```
XC      S_MODE,S_MODE
MVI     S_MODE2,S_IRUSR      Read search write read search
MVI     S_MODE3,S_IRUSR+S_IWGRP+S_IROTH+S_IXOTH
SPACE   ,
CALL    BPX4MKD,             Make a directory                +
      (BUFLNA,               Input: Pathname length          +
      BUFFERA,               Input: Pathname                  +
      S_MODE,                 Input: BPXYMODE and BPXYFTYP     +
      RETVAL,                 Return value: 0 or -1            +
      RETCODE,                Return code                      +
      RSNCODE),              Reason code                      +
      MF=(E,PLIST)           -----
```

## BPX4MKN (mknod) example

The following code creates a FIFO (pipe) named /u/fifos/fifo1 and user read-write, group read, other read permissions. For the callable service, see “mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file” on page 427. For the data structure, see “BPXYFTYP — File type definitions” on page 1078 and “BPXYMODE — Map the mode constants ” on page 1106. AMODE 31 callers use “BPX1MKN (mknod) example” on page 1261.

```
MVC     BUFFERA(14),=CL14'/u/fifos/fifo1'
MVC     BUFLNA,=F'14'
XC      S_MODE,S_MODE
MVI     S_TYPE,FT_FIFO       First in - first out
MVI     S_MODE2,S_IRUSR      Read write read read
MVI     S_MODE3,S_IWUSR+S_IRGRP+S_IROTH
SPACE   ,
CALL    BPX4MKN,             Create FIFO or char special file +
      (BUFLNA,               Input: Pathname length          +
      BUFFERA,               Input: Pathname                  +
      S_MODE,                 Input: BPXYMODE and BPXYFTYP     +
      =A(0),                  Input: Device id not used here   +
      RETVAL,                 Return value: 0 or -1            +
      RETCODE,                Return code                      +
      RSNCODE),              Reason code                      +
      MF=(E,PLIST)           -----
```

## BPX4MMI (\_\_map\_init) example

The following code creates a shared memory map with 10 map blocks each with a size of 1 meg. For the callable service, see “\_\_map\_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area” on page 412. For the data structure, see “BPXYMMG — Map interface for \_\_map\_init and \_\_map\_service ” on page 1101. AMODE 31 callers use “BPX1MMI (\_\_map\_init) example” on page 1261.

```
LA      R2,INITPARM           Set address of init parm list
STG     R2,INITADDR
USING   _MMG_INIT_PARM,R2
XC      _MMG_INIT_PARM(_MMG_INIT_PARM_LEN),_MMG_INIT_PARM
L       R1,=F'10'             Map area to contain 10 blocks
ST      R1,_MMG_NUMBLKS       *
L       R1,=F'1'              Each block is to be 1 meg in size
ST      R1,_MMG_MEGSPERBLK    *
SPACE   ,
CALL    BPX4MMI,              +
      (=A(MMG_INIT),          Input: Function code          +
      INITADDR,                Input: __map_init parameter list +
      RETVAL,                  Return value: 0, -1              +
      RETCODE,                 Return code                      +
      RSNCODE),               Reason code                      +
      MF=(E,PLIST)           -----
```



## BPX4MMP (mmap) example

The following code changes the protection of a memory mapped area. For the callable service, see “mmap (BPX1MMP, BPX4MMP) — Map pages of memory” on page 435. AMODE 31 callers use “BPX1MMP (mmap) example” on page 1262.

MVC	FILEDESC,...	File descriptor	
SPACE	,		
CALL	BPX4MMP,	map pages of memory	+
	(MAP_ADDRESS,	Input: address of mapped area	+
	MAP_LENGTH,	Input: area length	+
	=A(MAP_PRIVATE),	Input: Map type	+
	FILEDESC,	Input: File descriptor	+
	=AD(0),	Input: File offset	+
	RETURNEDADDRESS,	Output: value mapped address	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4MMS (\_\_map\_service) example

The following code creates three new data blocks within a shared memory map. For the callable service, see “\_\_map\_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services” on page 416. For the data structure, see “BPXYMMG — Map interface for \_map\_init and \_map\_service” on page 1101. AMODE 31 callers use “BPX1MMS (\_\_map\_service) example” on page 1262.

LA	R3,SRVCPARM	Set address of init parm list	
STG	R3,SRVCADDR		
USING	_MMG_SERVICE_PARM,R3		
XC	_MMG_SERVICE_PARM(_MMG_SERVICE_PARM_LEN),_MMG_SERVICE_PARM		
LA	R4,_MMG_NEWBLOCK	Request that a block be created	
STH	R4,_MMG_SERVICETYPE		
LA	R3,_MMG_SERVICE_PARM_LEN(R3)	Bump to next entry	
STH	R4,_MMG_SERVICETYPE	Create a second block	
LA	R3,_MMG_SERVICE_PARM_LEN(R3)	Bump to next entry	
STH	R4,_MMG_SERVICETYPE	Create the third block	
SPACE	,		
CALL	BPX4MMS,		+
	(=A(MMG_SERVICE),	Input: Function code	+
	SRVCADDR,	Input: __map_service parm list	+
	=F'3',	Input: Three requests to process	+
	_MMG_MAPTOKEN,	Map area token from INIT call	+
	RETVAL,	Return value: 0, -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4MNA (mknodat) example

The following code creates the file for the file identified by Pathname within the directory represented by the file descriptor FILEFD. Prior to this code excerpt, FILEFD would have been to the file descriptor of the specified directory or to AT\_FDCWD to indicate the current directory. For the callable service, see “mknodat (BPX1MNA, BPX4MNA) — Make a directory, a FIFO, a character special, or a regular file relative to a directory file descriptor” on page 431. For the data structure, see “BPXYFTYP — File type definitions” on page 1078 and “BPXYMODE — Map the mode constants” on page 1106. AMODE 31 callers use “BPX1MNA (mknodat) example” on page 1262.

MVC	BUFFERA(14),=CL14'/u/fifos/fifo1'	
MVC	BUFLINA,=F'14'	
XC	S_MODE,S_MODE	
MVI	S_TYPE,FT_FIFO	First in - first out
MVI	S_MODE2,S_IRUSR	Read write read read
MVI	S_MODE3,S_IWUSR+S_IRGRP+S_IROTH	
SPACE	,	
CALL	BPX4MNA,	Create FIFO or char special file +

## BPX4MNT (\_\_mount) example

(BUFLNA,	Input: Pathname length	+
BUFFERA,	Input: Pathname	+
S_MODE,	Input: BPXYMODE and BPXYFTYP	+
=A(0),	Input: Device id not used here	+
FILEDESC	Input: Directory file descriptor	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX4MNT (\_\_mount) example

The following code requests that the file system \_\_mount the system file and ready it for use. The file system name and mount parameters are encoded into the various fields in the MNTE. See “mount (BPX1MNT) — Make a file system available” on page 444. AMODE 31 callers use “BPX2MNT (\_\_mount) example” on page 1263.

LA	R14,MNTEH	R14->MNTEH and MNTE	
L	R15,MNTEL	R15 = Length of MNTEH and MNTE	
XR	R0,R0	Dummy 2nd operand	
XR	R1,R1	Pad=null, length=0	
MVCL	R14,R0	Null out MNTEH and MNTE	
MVC	MNTEHID,=CL4'MNT2'	Version indicator	
MVC	MNTEHLEN,=A(MNTE#LENGTH)	Length of MNTE	
MVC	MNTENTFSTNAME(08),=CL08'HFS'	HFS type name	
MVC	MNTENTFSNAME(44),=CL44'TESTLIB.FILESYS1'	Filesystem	
MVC	MNTENTMOUNTPOINT(02),=CL02'/u'	Mount point	
MVC	MNTENTPATHLEN,=F'2'		
MVC	MNTENTFSMODE4,=A(MNTENTFSMODERDONLY)	Filesystem mode	
CALL	BPX4MNT,	Ready a file system for use	+
	(MNTEL,	Input: MNTE length (hdr + body)	+
	MNTEH,	Input: MNTE	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4MP (mvspause) example

The following code places this thread into an MVS WAIT, to be terminated when a user ECB specified on a prior MVSpauseInit call is POSTed. The MVS WAIT is also terminated if a signal occurs. For the callable service, see “mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals” on page 479. AMODE 31 callers use “BPX1MP (mvspause) example” on page 1263.

CALL	BPX4MP,	MVS Pause	+
	(RETVAL,	Return value: 0, -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4MPC (mvsprocclp) example

The following code causes all z/OS UNIX-related resources to be released for this thread, and if this is the last thread in the process, for the process. For the callable service, see “mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources” on page 484. For the data structure, see “BPXYWAST — Map the wait status word” on page 1172. AMODE 31 callers use “BPX1MPC (mvsprocclp) examples” on page 1264.

XC	WAST(WAST#LENGTH),WAST		
MVI	WASTEXITCODE,57	User defined exit code	
SPACE	,		
CALL	BPX4MPC,	MVS Process cleanup	+
	(WAST,	Input: Ending status code 0-255	+
	RETVAL,	Return value: 0, -1 or 1	+
	RETCODE,	Return code	+

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4MPI (mvspauseinit) example

The following code prepares the thread for a subsequent MVSpause invocation. A list of Event Control Block addresses is passed to the system with the last address having the high order bit on. This syscall will use the first ECB pointed to from the list as the signal ECB, therefore at least one ECB address must be passed to the system. For the callable service, see “mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals” on page 479. AMODE 31 callers use “BPX1MPI (mvspauseinit) example” on page 1265.

	LA	R15,BUFFERA	Load address of ECB address list	
	STG	R15,BUFA	Save address for future parameter	
*			to be passed to BPX4MPI	
	SR	R15,R15	Clear R15	
	ST	R15,ECB01	Clear ECB01	
	ST	R15,ECB02	Clear ECB02	
	LA	R15,ECB01	Load address of first ECB	
	ST	R15,BUFFERA	Save ECB address in list of	
*			pointers	
	LA	R15,ECB02	Load address of second ECB	
*	ST	R15,BUFFERA+4	Save ECB address in list of	
			pointers	
	OI	BUFFERA+4,X'80'	Denote end of ECB pointers	
	SPACE	,		
	CALL	BPX4MPI,	MVS Pause initialize	+
		(BUFA,	Input ->list of ECB@, x'80' ended	+
		RETVAl,	Return value: 0, -1	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		MF=(E,PLIST)	-----	

## BPX4MPR (mprotect) example

The following code changes the protection of a memory mapped area. For the callable service, see “mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping” on page 452. AMODE 31 callers use “BPX1MPR (mprotect) example” on page 1265.

CALL	BPX4MPR,	set protection of a mapped area	+
	(MAP_ADDRESS,	Input: address of mapped area	+
	MAP_LENGTH,	Input: area length	+
	=A(PROT_READ),	Input: Protection options	+
	RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4MSD (mvsunsigsetup) example

The following code detaches the invoker from being able to catch signals. For the callable service, see “mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup” on page 495. AMODE 31 callers use “BPX1MSD (mvsunsigsetup) example” on page 1265.

CALL	BPX4MSD,	Reregister MVS signals, this task	+
	(SIRTNA,	Signal interface routine address	+
	USERWORD,	User data	+
	INTMASK,	Default override signal set	+
	TERMMASK,	Default terminate signal set	+
	RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4MSS (mvssigsetup) example

The following code allows the invoker to catch signals. For the callable service, see “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 487. AMODE 31 callers use “BPX1MSS (mvssigsetup) example” on page 1266.

```
* Each bit of the mask represents a signal 1-64.
MVC  INTMASK(8),=XL8'F000000000000000'      Default sig 1-4
MVC  TERMMASK(8),=XL8'F000000000000000'      Terminate sig 1-4
LA   R15,BUFFERA
STG  R15,USERWORD
SPACE ,
CALL BPX4MSS,                                Register MVS signals, this task +
      (=AD(SIRTN),                          Input: Signal interrupt routine +
      USERWORD,                             Input: User data +
      INTMASK,                              Input: Default override signals +
      TERMMASK,                             Input: Default terminate signals +
      RETVAL,                               Return value: 0 or -1 +
      RETCODE,                              Return code +
      RSNCODE),                             Reason code +
      MF=(E,PLIST)                          -----
```

## BPX4MSY (msync) example

The following code causes the file associated with this mapped area to be updated with the contents of storage. For the callable service, see “msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage” on page 469. AMODE 31 callers use “BPX1MSY (msync) example” on page 1266.

```
MVC  FILEDESC,..                            File descriptor
SPACE ,
CALL BPX4MSY,                                synchronize memory with storage +
      (MAP_ADDRESS,                         Input: address of mapped area +
      MAP_LENGTH,                           Input: area length +
      =A(MS_SYNC),                         Input: sync options +
      RETVAL,                               Return value: 0 or -1 +
      RETCODE,                              Return code +
      RSNCODE),                             Reason code +
      MF=(E,PLIST)                          -----
```

## BPX4MUN (munmap) example

The following code causes a mapped area to be unmapped. For the callable service, see “munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses” on page 473. AMODE 31 callers use “BPX1MUN (munmap) example” on page 1266.

```
CALL BPX4MUN,                                unmap previously mapped addresses +
      (MAP_ADDRESS,                         Input: address of mapped area +
      MAP_LENGTH,                           Input: area length +
      RETVAL,                               Return value: 0 or -1 +
      RETCODE,                              Return code +
      RSNCODE),                             Reason code +
      MF=(E,PLIST)                          -----
```

## BPX4NIC (nice) example

The following code increases the priority value of the calling process by 1. For the callable service, see “nice (BPX1NIC, BPX4NIC) — Change the nice value of a process” on page 497. AMODE 31 callers use “BPX1NIC (nice) example” on page 1266.

```
MVC  INCR,=F'1'                             Increase priority by 1
SPACE ,
CALL BPX4NIC,                                Change priority value +
      (INCR,                               Input: Priority change value +
      RETVAL,                               Return value: new nice value or -1+
```

	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX4OPD (opendir) example

The following code opens directory /etc/passwd so that it can be read by readdir. For the callable service, see [“opendir \(BPX1OPD, BPX4OPD\) — Open a directory”](#) on page 523. AMODE 31 callers use [“BPX1OPD \(opendir\) example”](#) on page 1267.

MVC	BUFLINA,=F'11'		
MVC	BUFFERA(11),=CL11' /etc/passwd'		
SPACE	,		
CALL	BPX4OPD,	Open a directory	+
	(BUFLINA,	Input: Directory name length	+
	BUFFERA,	Input: Directory name	+
	RETVAL,	Return value:-1 or directory f.d.	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BL	PSEUDO	Branch if negative (-1 = failure)	
ST	R15,DIRECTDES	Store the directory descriptor	

## BPX4OPN (open) example

The following code opens file usr/inv/nov.d with user read-write, group read and other read. A file descriptor (FILEDESC) is returned. For the callable service, see [“open \(BPX1OPN, BPX4OPN\) — Open a file”](#) on page 511. For the data structure, see [“BPXYOPNF — Map flag values for open”](#) on page 1113, [“BPXYMODE — Map the mode constants”](#) on page 1106, and [“BPXYFTYP — File type definitions”](#) on page 1078. AMODE 31 callers use [“BPX1OPN \(open\) example”](#) on page 1267.

MVC	BUFFERA(13),=CL13'usr/inv/nov.d'		
MVC	BUFLINA,=F'13'		
XC	S_MODE,S_MODE		
MVI	S_MODE2,S_IRUSR	User read/write, group read,	
MVI	S_MODE3,S_IWUSR+S_IRGRP+S_IROTH	other read	
XC	O_FLAGS(OPNF#LENGTH),O_FLAGS		
MVI	O_FLAGS4,O_CREAT+O_RDWR	Create, open for read and write	
SPACE	,		
CALL	BPX4OPN,	Open a file	+
	(BUFLINA,	Input: Pathname length	+
	BUFFERA,	Input: Pathname	+
	O_FLAGS,	Input: Access	BPXYOPNF +
	S_MODE,	Input: Mode	BPXYMODE, BPXYFTYP +
	RETVAL,	Return value:-1 or file descriptor	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test RETVAL	
BL	PSEUDO	Branch if negative (-1 = failure)	
ST	R15,FILEDESC	Store the file descriptor	

## BPX4OPS (openstat) example

The following code opens file usr/inv/nov.d. with user read-write, group read and other read, and obtains status about the file. A file descriptor (FILEDESC) is returned. For the callable service, see [“openstat \(BPX2OPN, BPX4OPS\) — Open a file and obtain status information”](#) on page 526. For the data structures, see [“BPXYOPNF — Map flag values for open”](#) on page 1113, [“BPXYSTAT — Map the response structure for stat”](#) on page 1162, [“BPXYMODE — Map the mode constants”](#) on page 1106, and

## BPX4OPT (getsockopt or setsockopt) example

“BPXYFTYP — File type definitions” on page 1078. AMODE 31 callers use “BPX2OPN (openstat) example” on page 1267.

```
MVC  BUFFERA(13),=CL13'usr/inv/nov.d'
MVC  BUFLINA,=F'13'
XC   S_MODE,S_MODE
MVI  S_MODE2,S_IRUSR      User read/write, group read,
MVI  S_MODE3,S_IWUSR+S_IRGRP+S_IROTH      other read
XC   O_FLAGS(OPNF#LENGTH),O_FLAGS
MVI  O_FLAGS4,O_CREAT+O_RDWR Create, open for read and write
SPACE ,
CALL  BPX4OPS,            Open a file and get status          +
      (BUFLINA,          Input: Pathname length              +
      BUFFERA,           Input: Pathname                      +
      O_FLAGS,           Input: Access      BPXYOPNF          +
      S_MODE,            Input: Mode      BPXYMODE, BPXYFTYP +
      STATL,             Input: Length of buffer needed      +
      STAT,              Buffer, BPXYSTAT                    +
      RETVAL,            Return value:-1 or file descriptor+
      RETCODE,           Return code                          +
      RSNCODE),          Reason code                          +
      MF=(E,PLIST)      -----
ICM  R15,B'1111',RETVAL  Test RETVAL
BL   PSEUDO              Branch if negative (-1 = failure)
ST   R15,FILEDESC        Store the file descriptor
```

## BPX4OPT (getsockopt or setsockopt) example

The following code gets and then sets socket options. SOCKDESC was returned on a previous call to BPX4SOC. For the callable service, see “getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket” on page 322. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1OPT (getsockopt or setsockopt) example” on page 1268.

```
MVC  BUFLINA,=A(L'BUFFERA)
CALL  BPX4OPT,            Get socket options                  +
      (SOCKDESC,          Input: Socket Descriptor          +
      =A(SOCK#0PTOPTGETSOCKOPT), Input: Indicate Get socket +
      SOCK#SO_SOCKET,     Input: Level                      +
      SOCK#SO_TYPE,       Input: Option name                +
      BUFLINA,            Input: Length - option value      +
      BUFFERA,            Input: Option value               +
      RETVAL,             Return value: 0 or -1              +
      RETCODE,            Return code                        +
      RSNCODE),          Reason code                        +
      MF=(E,PLIST)      -----
SPACE ,
MVC  BUFLINA,=A(4)        SO_00BINLINE has length=4
CALL  BPX4OPT,            Set socket options                  +
      (SOCKDESC,          Input: Socket Descriptor          +
      =A(SOCK#0PTOPTSETSOCKOPT), Input: Indicate set socket +
      SOCK#SO_SOCKET,     Input: Level                      +
      SOCK#SO_TYPE,       Input: Option name                +
      BUFLINA,            Input: Length - option value      +
      SOCK#SO_00BINLINE,  Input: Option value               +
      RETVAL,             Return value: 0 or -1              +
      RETCODE,            Return code                        +
      RSNCODE),          Reason code                        +
      MF=(E,PLIST)      -----
```

## BPX4PAF (\_\_pid\_affinity) example

The following code will add your PID to the target process' affinity list. For the callable service, see “\_\_pid\_affinity (BPX1PAF, BPX4PAF) — Add or delete an entry in a process's affinity list” on page 548. AMODE 31 callers use “BPX1PAF (\_\_pid\_affinity) example” on page 1268.

```
*      MVC  TARPID,....    PID of target
*      MVC  SIGPID,....    PID of this routine
CALL  BPX4PAF,            +
```

(=A(PAF_ADD_PID#),	Function code (add entry)	+
TARPID,	PID of target	+
SIGPID,	PID to receive signal	+
=A(SIGUSR1#),	signal to be generated	+
RETVAl,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4PAS (pause) example

The following code suspends execution of the invoker's thread until a signal is delivered. For the callable service, see [“pause \(BPX1PAS, BPX4PAS\) — Suspend a process pending a signal”](#) on page 539. AMODE 31 callers use [“BPX1PAS \(pause\) example”](#) on page 1269.

CALL BPX4PAS,	Suspend execution	+
(RETVAl,	Return value: -1 or not return	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4PCF (pathconf) example

The following code extracts the current value for the configurable maximum number of bytes in a file name associated with /usr/inv/network.t. For the callable service, see [“pathconf \(BPX1PCF, BPX4PCF\) — Determine configurable path name variables using a path name”](#) on page 535. For the data structure, see [“BPXYPCF — Command values for pathconf and pathconf”](#) on page 1114. AMODE 31 callers use [“BPX1PCF \(pathconf\) example”](#) on page 1269.

MVC BUFFERA(18),=CL18'/usr/inv/network.t'		
MVC BUFLINA,=F'18'		
SPACE ,		
CALL BPX4PCF,	Get configurable pathname variable+	
(BUFLINA,	Input: Pathname length	+
BUFFERA,	Input: Pathname	+
=A(PC_NAME_MAX),	Input: Options BPXYPCF	+
RETVAl,	Return value: 0, -1 or variable	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4PCT (pfsctl) example

The following code conveys a command to a Physical File System named ACMEFILE. ACMEFILE doesn't really exist; to actually run this example you would need a real PFS product that supports this function. For the callable service, see [“pfsctl \(BPX1PCT, BPX4PCT\) — Physical file system control”](#) on page 541. AMODE 31 callers use [“BPX1PCT \(pfsctl\) example”](#) on page 1269.

MVC FSTYPE(8),=CL08'ACMEFILE'		
MVC BUFLINA,=F'25'		
MVC BUFFERA(25),=CL25'COMPRESS(ON) CONVERT(OFF)'		
MVC COMMAND,=F'123'	PFS product defined command	
SPACE ,		
CALL BPX4PCT,	PFS Control	+
(FSTYPE,	Input: PFS Type Name	+
COMMAND,	Input: Command	+
BUFLINA,	Input: Argument length	+
BUFFERA,	Input/Output: Argument buffer	+
RETVAl,	Return value: product defined	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4PIP (pipe) example

The following code creates a pipe. For the callable service, see “[pipe \(BPX1PIP, BPX4PIP\) — Create an unnamed pipe](#)” on page 551. AMODE 31 callers use “[BPX1PIP \(pipe\) example](#)” on page 1270.

```
CALL BPX4PIP,          Create a pipe          +
    (READFD,          Output: Read file descriptor +
    WRITEFD,          Output: Write file descriptor +
    RETVAL,           Return value: 0 or -1        +
    RETCODE,          Return code                 +
    RSNCODE),         Reason code                 +
    MF=(E,PLIST)      -----
```

## BPX4PI2 (pipe2) example

The following code creates a **pipe2** instance. For the callable service, use “[pipe2 \(BPX1PI2, BPX4PI2\) — Create a new pipe instance](#)” on page 553. AMODE 32 callers use the . “[BPX1PI2 \(pipe2\) example](#)” on page 1270.

```
CALL BPX4PI2,          Create a pipe2 +
    (READFD,          Output: Read file descriptor +
    WRITEFD,          Output: Write file descriptor +
    FLAGS,            Input: Flags +
    RETVAL,           Return value: 0 or -1 +
    RETCODE,          Return code +
    RSNCODE),         Reason code +
    VL,MF=(E,PLIST)
```

## BPX4POE (\_\_poe) example

The following code registers a socket (SOCKDESC) as the process scope port of entry. SOCKDESC was returned previously from a call to either BPX4SOC or BPX4ACP. For the callable service, see “[\\_\\_poe\(\) \(BPX1POE, BPX4POE\) — Port of entry information](#)” on page 559. For the data structure, see “[BPXYPOE — Map poe syscall parameters](#)” on page 1123. AMODE 31 callers use “[BPX1POE \(\\_\\_poe\) example](#)” on page 1270.

```
MVC POEOPTIONS,=A(POE#SCOPEPROCESS)
MVC POENTRYTYPE,=A(POE#ENTRYSOCKET)
MVC POENTRYLEN,=A(POE#ENTRYSOCKETLEN)
LA  R15,SOCKDESC
STG R15,POENTRYPTR64
CALL BPX4POE,          Port of Entry registration  +
    (=A(POE#LEN),      Input: Length of poe structure +
    POE,               Input: mapped by BPXYPOE      +
    RETVAL,            Return value: 0 or -1          +
    RETCODE,           Return code                   +
    RSNCODE),          Reason code                   +
    MF=(E,PLIST)      -----
```

## BPX4POL (poll) example

The following code issues a poll. For the callable service, see “[poll \(BPX1POL, BPX4POL\) — Monitor activity on file descriptors and message queues](#)” on page 564. For the data structure, see “[BPXYPOLL — Map poll syscall parameters](#)” on page 1124. AMODE 31 callers use “[BPX1POL \(poll\) example](#)” on page 1270.

```
*      LA  R15,BUFFERA
      USING POLLFD,R15
      STG R15,BUFA          ->BPXYPOLL structure
      MVC POLLHFD(4),file_descriptor_number2
      MVI POLLEVENTS,0
      MVI POLLEVENTS+1,POLLERDNORM
      A   R15,=A(POLLFD#LENGTH)
```



```

*      MVC  POLLHFD(4),file_descriptor_number1
      MVI   POLLEVENTS,0
      MVI   POLLEVENTS+1,POLLEWRNORM
      SPACE ,
      CALL  BPX4POL,          Create a pipe          +
      (BUFA,                  Input: address of BPXPOLL      +
      =A(2),                  Input: number of BPXPOLL structs +
      =A(0),                  Input: -1, 0, milliseconds      +
      RETVAL,                 Return value: 0 or -1          +
      RETCODE,                Return code                  +
      RSNCODE),               Reason code                  +
      MF=(E,PLIST)           -----

```

## BPX4PRC (prctl) example

The following code creates a **prctl** instance. For the callable service, use “prctl (BPX1PRC, BPX4PRC) — Managing processes and threads” on page 568. AMODE31 callers use the “BPX1PRC (prctl) example” on page 1271.

1. The following code sets the name of the calling thread to My job thread 1.

```

      MVC   NAME(15),=CL15'My job thread 1'
      MVC   NAMELEN,=F'15'
      SPACE ,
      CALL  BPX4PRC,          Manage processes and threads +
      (=A(PR_SET_NAME),      Input: prctl constant in BPXYCONS +
      NAMELEN,               Input: Thread name length      +
      NAME,                  Input: Thread name            +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      RETVAL,                Return value: 0 or -1          +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)

```

2. The following code retrieves the name of the calling thread.

```

      CALL  BPX4PRC,          Manage processes and threads +
      (=A(PR_GET_NAME),      Input: prctl constant in BPXYCONS +
      NAMELEN,               Output: Thread name length      +
      NAME,                  Output: Thread name            +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      =A(0),                 Unused for this request type    +
      RETVAL,                Return value: 0 or -1          +
      RETCODE,               Return code                    +
      RSNCODE),              Reason code                    +
      VL,MF=(E,PLIST)

```

```

      .
      .
      .

```

## BPX4PRL (prlimit) example

The following code sets the address space resource limits to RLIM\_INFINITY for the specified target PID. The new limits are set in the RLIMIT structure and passed as input. This specific syscall invocation shows only the new limit (set function) that was requested and the old limit parameter is omitted. Alternatively, the prlimit invoker can pass in only the previous limit structure to be filled in by the service or pass in both previous and new limits. For the callable service, see “prlimit (BPX1PRL, BPX4PRL) — Get and set resource limits for specified processes” on page 574. For the data structure, see “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142. AMODE 32 callers use “BPX1PRL (prlimit) example” on page 1271.

```

      XC     NEW_RLIM(RLIMIT#LENGTH),NEW_RLIM  Clear New limit
      LA     R5,NEW_RLIM
      USING  RLIMIT,R5

```

## BPX4PSI (pthread\_setintr) example

```
MVC  RLIM_CUR,=A(RLIM_INFINITY) Current limit
MVC  RLIM_MAX,=A(RLIM_INFINITY) Maximum limit
CALL  BPX4PRL,
      (TARGPID,          PID of target process
      =A(RLIMIT_AS),     Address space resource
      NEW_RLIM,          New Limit (Set)
      0,                 Old Limit (Get)
      RETVAL,
      RETCODE,
      RSNCODE),
      MF=(E,PLIST)
L     R15,RETVAL         Load return value
C     R15,=F'-1'         Test for -1 return
BE    PSEUDO             Branch on error
```

## BPX4PSI (pthread\_setintr) example

The following code sets the interruption type of the calling thread. For the callable service, see “[pthread\\_setintr \(BPX1PSI, BPX4PSI\) — Examine and change the interrupt state](#)” on page 610. For the data structure, see “[BPXYCONS — Constants used by services](#)” on page 1062. AMODE 31 callers use “[BPX1PSI \(pthread\\_setintr\) example](#)” on page 1272.

```
CALL  BPX4PSI,           Examine and change interrupt state+
      (INTRSTATE,        Input: Interrupt state  BPXYCONS +
      RETVAL,            Return value: 0 or -1
      RETCODE,           Return code
      RSNCODE),          Reason code
      MF=(E,PLIST)       -----
```

## BPX4PST (pthread\_setintrtype) example

The following code sets the interruption type of the calling thread and returns the previous interruption type. For the callable service, see “[pthread\\_setintrtype \(BPX1PST, BPX4PST\) — Examine and change the interrupt type](#)” on page 613. For the data structure, see “[BPXYCONS — Constants used by services](#)” on page 1062. AMODE 31 callers use “[BPX1PST \(pthread\\_setintrtype\) example](#)” on page 1272.

```
CALL  BPX4PST,           Examine and change interrupt type +
      (INTRTYPE,         Input: Interrupt type  BPXYCONS +
      RETVAL,            Return value: 0 or -1
      RETCODE,           Return code
      RSNCODE),          Reason code
      MF=(E,PLIST)       -----
```

## BPX4PTB (pthread\_cancel) example

The following code generates a cancelation request for the target thread (THID). For the callable service, see “[pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread](#)” on page 579. AMODE 31 callers use “[BPX1PTB \(pthread\\_cancel\) example](#)” on page 1272.

```
CALL  BPX4PTB,           pthread_cancel
      (THID,             Input: Thread ID
      RETVAL,            Return Value: 0, -1, or Buf length+
      RETCODE,           Return code
      RSNCODE),          Reason code
      MF=(E,PLIST)       -----
```

## BPX4PTC (pthread\_create) example

The following code creates a new thread. For the callable service, see “[pthread\\_cancel \(BPX1PTB, BPX4PTB\) — Cancel a thread](#)” on page 579. For the data structure, see “[BPXYPTAT — Map attributes](#)”

for `pthread_exit_and_get` on page 1127. AMODE 31 callers use [“BPX1PTC \(pthread\\_create\) example”](#) on page 1272.

LA	R15,BUFFERA	Work area	
STG	R15,BUFA	->above	
LA	R15,PTAT	Area mapped by BPXYPTAT	
STG	R15,PTATA	->above	
MVC	PTATEYE,=C'BPXYPTAT'	Set the eye-catcher	
MVC	PTATLENGTH,=A(PATUSEROFFVAL)	Length of structure	
MVC	PTATSYSOFFSET,=A(PATSYSOFFVAL)	Sys attr offset	
MVC	PTATSYSLLENGTH,=A(PATSYSLLENVAL)	Sys attr length	
MVC	PTATUSEROFFSET,=A(0)	User attr offset	
MVC	PTATUSERLENGTH,=A(0)	User attr length	
LOAD	EP=INITRTN	Get address of Init Rtn	
STG	R0,INITRTNA		
SPACE	,		
CALL	BPX4PTC,		+
	(INITRTNA,	Input: Init routine address	+
	BUFA,	Input: Work area address	+
	PTATA,	Input: Attr area Address BPXYPTAT	+
	THID,	Thread ID, if Return value = 0	+
	RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4PTD (pthread\_detach) example

The following code detaches a thread (THID) in the calling process. For the callable service, see [“pthread\\_detach \(BPX1PTD, BPX4PTD\) — Detach a thread”](#) on page 586. AMODE 31 callers use [“BPX1PTD \(pthread\\_detach\) example”](#) on page 1273.

CALL	BPX4PTD,	pthread_detach	+
	(THID,	Input: Thread ID	+
	RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4PTI (pthread\_testintr) example

The following code causes a cancellation point. For the callable service, see [“pthread\\_testintr \(BPX1PTI, BPX4PTI\) — Cause a cancellation point to occur”](#) on page 618. AMODE 31 callers use [“BPX1PTI \(pthread\\_testintr\) example”](#) on page 1273.

CALL	BPX4PTI,	Cause an interrupt point to occur	+
	(RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4PTJ (pthread\_join) example

The following code gets the termination status of a specified thread (THID). For the callable service, see [“pthread\\_join \(BPX1PTJ, BPX4PTJ\) — Wait on a thread”](#) on page 592. AMODE 31 callers use [“BPX1PTJ \(pthread\\_join\) example”](#) on page 1273.

CALL	BPX4PTJ,	pthread_join	+
	(THID,	Input: Thread ID	+
	=AD(0),	Input: ->Status Field or 0	+
	RETVAl,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4PTK (pthread\_kill) example

The following code sends a signal to a specified thread (THID). For the callable service, see “pthread\_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread” on page 595. For the data structure, see “BPXYSIGH — Signal constants” on page 1148. AMODE 31 callers use “BPX1PTK (pthread\_kill) example” on page 1274.

```

MVC  SIGNAL,=A(SIGALRM#)      Input: SIGALRM      BPXYSIGH
MVC  SIGNALOPTIONS,=XL4'00000000' Input: Signal options
CALL BPX4PTK,                  pthread_kill      +
      (THID,                   Input: Thread ID    +
      SIGNAL,                  Input: Signal or 0    BPXYSIGH +
      SIGNALOPTIONS,           Input: Signal options +
      RETVAL,                  Return value: 0 or -1 +
      RETCODE,                 Return code         +
      RSNCODE),                Reason code         +
      MF=(E,PLIST)             -----

```

## BPX4PTQ (pthread\_quiesce) example

The following code terminates all other pthreads in the caller's process. For the callable service, see “pthread\_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process” on page 598. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 31 callers use “BPX1PTQ (pthread\_quiesce) example” on page 1274.

```

CALL BPX4PTQ,                  pthread_quiesce      +
      (=A(QUIESCE_TERM),       Input: Quiesce type    BPXYCONS +
      =AD(0),                  Input: User data - Catch data PPSD+
      RETVAL,                  Return value: 0 or -1 +
      RETCODE,                 Return code         +
      RSNCODE),                Reason code         +
      MF=(E,PLIST)             -----

```

## BPX4PTR (ptrace) example

The following code enables a process (PROCID) to be debugged with ptrace. For the callable service, see “ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging” on page 620. For the data structure, see “BPXYPTRC — Map parameters for ptrace” on page 1127. AMODE 31 callers use “BPX1PTR (ptrace) example” on page 1274.

```

*      MVC  PROCID, Process ID from fork
      SPACE ,
      CALL BPX4PTR,             Debug another process +
          (=A(PT_ATTACH),       Input: Request      BPXYPTRC +
          PROCID,               Input: Process ID    +
          =AD(0),               Input: Address       +
          =AD(0),               Input: Data          +
          =AD(0),               Input: Buffer         +
          RETVAL,               Return value: 0, -1, or Request +
          RETCODE,              Return code         +
          RSNCODE),             Reason code         +
          MF=(E,PLIST)          -----

```

## BPX4PRL (prlimit) example

The following code sets the address space resource limits to RLIM\_INFINITY for the specified target PID. The new limits are set in the RLIMIT structure and passed as input. This specific syscall invocation shows only the new limit (set function) that was requested and the old limit parameter is omitted. Alternatively, the prlimit invoker can pass in only the previous limit structure to be filled in by the service or pass in both previous and new limits. For the callable service, see “prlimit (BPX1PRL, BPX4PRL) — Get and set resource limits for specified processes” on page 574. For the data structure, see “BPXYRLIM — Map the

rlimit, rusage, and timeval structures” on page 1142. AMODE 32 callers use “BPX1PRL (prlimit) example” on page 1271.

```

XC    NEW_RLIM(RLIMIT#LENGTH),NEW_RLIM  Clear New limit
LA    R5,NEW_RLIM
USING RLIMIT,R5
MVC   RLIM_CUR,=A(RLIM_INFINITY) Current limit
MVC   RLIM_MAX,=A(RLIM_INFINITY) Maximum limit
CALL  BPX4PRL,
      (TARGPID,          PID of target process      +
      =A(RLIMIT_AS),     Address space resource     +
      NEW_RLIM,          New Limit (Set)            +
      0,                 Old Limit (Get)            +
      RETVAL,            +
      RETCODE,           +
      RSNCODE),          +
      MF=(E,PLIST)
L      R15,RETVAL        Load return value
C      R15,=F'-1'        Test for -1 return
BE     PSEUDO            Branch on error

```

## BPX4PTS (pthread\_self) example

The following code gets the thread ID of the calling thread. For the callable service, see “pthread\_self (BPX1PTS, BPX4PTS) — Query the thread ID ” on page 609. AMODE 31 callers use “BPX1PTS (pthread\_self) example” on page 1274.

```

CALL  BPX4PTS,          pthread_self      +
      (THID),           Output: Thread ID +
      MF=(E,PLIST)      -----

```

## BPX4PTT (pthread\_tag\_np) example

The following code updates the pthread tag. For the callable service, see “pthread\_tag\_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data ” on page 615. AMODE 31 callers use “BPX1PTT (pthread\_tag\_np) example” on page 1275.

```

LA    R15,=CL30'UPDATING MONTH-END STATISTICS'
STG   R15,PT_NEWA
LA    R15,PT_OLD
STG   R15,PT_OLDA
CALL  BPX4PTT,          pthread_tag_np    +
      (=A(30),          Input: Length of New Tag +
      PT_NEWA,          Input: Address of New Tag +
      PT_OLDL,          Input: Length of Old Tag +
      PT_OLDA,          Input: Address to store Old Tag +
      RETVAL,           Return value: 0 or -1    +
      RETCODE,          Return code:            +
      RSNCODE),         Reason code:            +
      MF=(E,PLIST)      -----

```

## BPX4PTX (pthread\_exit\_and\_get) example

The following code terminates a thread and creates a new thread. For the callable service, see “pthread\_exit\_and\_get (BPX1PTX, BPX4PTX) — Exit and get a new thread ” on page 588. For the data structure, see “BPXYCONS — Constants used by services” on page 1062. AMODE 31 callers use “BPX1PTX (pthread\_exit\_and\_get) example ” on page 1275.

```

CALL  BPX4PTX,          pthread_exit_and_get +
      (STATFLD,         Input: Status field    +
      OPTIONS,          Input: Options field   +
      SIGNALREG,        Input: Signal registration usrdata+
      RETVAL,           Return value: 0 or -1  ->BPXYPTXL +
      RETCODE,          Return code           +

```

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4PVR (pivot\_root) example

The following code moves the root mount to the /usr/inv/new\_root/put\_old directory and makes /usr/inv/new\_root the new root mount. For the callable service, see “pivot\_root (BPX1PVR, BPX4PVR) — Change the root mount” on page 556. For the data structure, see “BPXYMNTE — Map response and element structure of w\_getmntent” on page 1104. AMODE 31 callers use “BPX1PVR (pivot\_root) example” on page 1275.

MVC	NEWROOT(18),=CL18'/usr/inv/new_root'		
MVC	NEWROOTLEN,=F'18'		
MVC	PUTOLD(26),=CL26'/usr/inv/new_root/put_old'		
MVC	PUTOLDLEN,=F'26'		
SPACE	,		
CALL	BPX4PVR,	Pivot Root	+
	(NEWROOTLEN,	Input: New Root Path length	+
	NEWROOT,	Input: New Root Pathname	+
	PUTOLDLEN,	Input: Put Old Path length	+
	PUTOLD,	Input: Put Old Pathname	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	VL,MF=(E,PLIST)	-----	

## BPX4PWD (\_\_passwd, \_\_passwd\_\_applid) example

The following code queries/changes the password of a given user ID. For the callable service, see “\_\_passwd, \_\_passwd\_\_applid (BPX1PWD, BPX4PWD) — Verify or change security information” on page 530. AMODE 31 callers use “BPX1PWD (\_\_passwd, \_\_passwd\_\_applid) example” on page 1276.

MVC	USERNLEN,=F'8'		
MVC	USERNAME(8),=CL8'Myuserid'		
MVC	OLDPASSLEN,=F'8'		
MVC	OLDPASS(8),=CL8'MyOldPwd'		
MVC	NEWPASSLEN,=F'8'		
MVC	NEWPASS(8),=CL8'MyNewPwd'		
SPACE	,		
CALL	BPX4PWD,	Query/change user ID password	+
	(USERNLEN,	Input: Length of user ID	+
	USERNAME,	Input: User ID	+
	OLDPASSLEN,	Input: Length of old password	+
	OLDPASS,	Input: Old password	+
	NEWPASSLEN,	Input: Length of new password	+
	NEWPASS,	Input: New password	+
	RETVAL,	Return value 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4QCT (msgctl) example

The following code removes the message queue from the system. For the callable service, see “msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations” on page 455. For the data structure, see “BPXYMSG — Map interprocess communication message queues” on page 1107. AMODE 31 callers use “BPX1QCT (msgctl) example” on page 1276.

CALL	BPX4QCT,	Message queue control (msgctl)	+
	(MSG_ID,	Input: MessageQueueID	+
	=A(IPC_RMID),	Input: Action to take	BPXYIPC
	=AD(0),	Input: ->MSQID_DS or 0	BPXYMSG
	RETVAL,	Return value: 0, -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4QDB (querydub) example

The following code obtains the dub status information for the current task. The status indicates whether the current task has already been dubbed, is ready to be dubbed, or cannot be dubbed as a process (or thread). AMODE 31 callers use “BPX1QDB (querydub) example” on page 1276.

```
CALL BPX4QDB,          Query DUB status for this task      +
    (RETVAL,          Return value: -1 or see BPXYCONS      +
    RETCODE,          Return code                          +
    RSNCODE),         Reason code                          +
    MF=(E,PLIST)      -----
```

## BPX4QGT (msgget) example

The following code creates a private message queue. For the callable service, see “msgget (BPX1QGT, BPX4QGT) — Create or find a message queue” on page 458. For the data structure, see “BPXYMSG — Map interprocess communication message queues” on page 1107. AMODE 31 callers use “BPX1QGT (msgget) example” on page 1276.

```
MVI S_TYPE,IPC_CREAT+IPC_EXCL      Error if exists
MVI S_MODE1,0                      Not used
MVI S_MODE2,S_IRUSR                All read and write permissions
MVI S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL BPX4QGT,                      Create a message queue      +
    (=A(IPC_PRIVATE),             Input: Key                  +
    S_MODE,                      Input: Creation flags BPXYMODE/IPC+
    RETVAL,                      Return value: -1 or msg ID +532200
    RETCODE,                    Return code                  +
    RSNCODE),                   Reason code                  +
    MF=(E,PLIST)                -----
SPACE ,
ICM R15,B'1111',RETVAL            Test return value
BNP PSEUDO                       Branch on msgget failure
ST  R15,MSG_ID                   Store MSG_ID associated with key
```

## BPX4QRC (msgrcv) example

The following code adds a message to the message queue identified by MSG\_ID. For the callable service, see “msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue” on page 462. For the data structure, see “BPXYMSG — Map interprocess communication message queues” on page 1107. AMODE 31 callers use “BPX1QRC (msgrcv) example” on page 1277.

```
LA R15,BUFFERA                    R15 -> Utility buffer
STG R15,BUFA
USING MSGBUF,R15
MVC MSG_TYPE(4),=A(0)
MVC BUFLINA(4),=A(MSG#LENGTH)
MVC FLAGS(4),=A(0)                Wait for message
DROP R15
SPACE ,
CALL BPX4QSN,                     Send a message (msgrcv)      +
    (MSG_ID,                    Input: MessageQueueID          +
    BUFA,                      Input: ->MSGBUF                BPXYMSG +
    PRIMARYALET,               Input: ALET of message buffer    +
    BUFLINA,                   Input: Length MSGBUF            +
    =AD(0),                   Input: Message Type              BPXYMSG +
    FLAGS,                    Input: Flags                    BPXYIPC +
    RETVAL,                   Return value: 0, -1              +
    RETCODE,                  Return code                      +
    RSNCODE),                 Reason code                      +
    MF=(E,PLIST)              -----
```

## BPX4QSE (quiesce) example

The following code quiesces file system TESTLIB.FILESYS1, making the files in it unavailable for use. For the callable service, see [“quiesce \(BPX1QSE, BPX4QSE\) — Quiesce a file system”](#) on page 650. AMODE 31 callers use [“BPX1QSE \(quiesce\) example”](#) on page 1277.

```

MVC  FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX4QSE,           Quiesce a file system          +
      (FSNAME,           Input: File system name (44 char) +
      RETVAL,            Return value: 0, -1, or 4         +
      RETCODE,           Return code                     +
      RSNCODE),          Reason code                     +
      MF=(E,PLIST)      -----

```

## BPX4QSN (msgsnd) example

The following code adds a message to the message queue identified by MSG\_ID. For the callable service, see [“msgsnd \(BPX1QSN, BPX4QSN\) — Send to a message queue”](#) on page 465. For the data structure, see [“BPXYMSG — Map interprocess communication message queues”](#) on page 1107. AMODE 31 callers use [“BPX1QSN \(msgsnd\) example”](#) on page 1277.

```

LA    R15,BUFFERA        R15 -> Utility buffer
STG   R15,BUFA
USING MSGBUF,R15
MVC   MSG_TYPE(4),=A(0)
MVC   MSG_MTEXT(11),=CL11'QSN MSG TEXT'
MVC   BUFLINA(4),=A(15)
MVC   FLAGS(4),=A(IPC_NOWAIT)  Don't wait on queue full
DROP  R15
SPACE ,
CALL  BPX4QSN,           Send a message (msgsnd)          +
      (MSG_ID,           Input: MessageQueueID           +
      BUFA,              Input: ->MSGBUF                 +
      PRIMARYALET,       Input: ALET of message buffer    +
      BUFLINA,           Input: Length MSGBUF            +
      FLAGS,             Input: Flags                    +
      RETVAL,            Return value: 0, -1              +
      RETCODE,           Return code                     +
      RSNCODE),          Reason code                     +
      MF=(E,PLIST)      -----

```

## BPX4RCV (recv) example

The following code issues a recv for a socket. SOCKDESC was returned previously from a call to either BPX4SOC or BPX4ACP. For the callable service, see [“recv \(BPX1RCV, BPX4RCV\) — Receive data on a socket and store it in a buffer”](#) on page 680. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYMSGF — Map the message flags”](#) on page 1107. AMODE 31 callers use [“BPX1RCV \(recv\) example”](#) on page 1278.

```

SPACE ,
CALL  BPX4RCV,           Receive data on from a socket    +
      (SOCKDESC,         Input: Socket Descriptor        +
      =A(L'BUFFERA),     Input: Length of input buffer    +
      BUFFERA,           Input: Address of input buffer    +
      PRIMARYALET,       Input: Alet of input buffer      +
      MSG_FLAGS,         Input: Flags                    +
      RETVAL,            Return value: 0 or -1            +
      RETCODE,           Return code                     +
      RSNCODE),          Reason code                     +
      MF=(E,PLIST)      -----

```



## BPX4RDD (readdir) example

The following code reads multiple name entries from the specified directory (DIRECTDES). For the callable service, see [“readdir \(BPX1RDD, BPX4RDD\) — Read an entry from a directory”](#) on page 656. For the data structure, see [“BPXYDIRE — Map directory entries for readdir”](#) on page 1074. AMODE 31 callers use [“BPX1RDD \(readdir\) example”](#) on page 1278.

```

MVC    DIRECTDES,...          Directory descriptor from opendir
LA      R15,BUFFERA
STG     R15,BUFA
MVC     BUFLINA,=F'1023'
CALL    BPX4RDD,              Read entries from a directory +
      (DIRECTDES,             Input: Directory file descriptor +
      BUFA,                   Output: ->buffer          BPXYDIRE +
      PRIMARYALET,           Input: buffer ALET          +
      BUFLINA,               Input: buffer size          +
      RETVAL,                Return value: 0, -1, entries read +
      RETCODE,               Return code                  +
      RSNCODE),              Reason code                  +
      MF=(E,PLIST)           -----

```

## BPX4RDL (readlink) example

The following code reads the contents of symbolic link /personnel/templink into the buffer provided. This will be the path name that was specified when the symbolic link was defined. For the callable service, see [“readlink \(BPX1RDL, BPX4RDL\) — Read the value of a symbolic link”](#) on page 666. AMODE 31 callers use [“BPX1RDL \(readlink\) example”](#) on page 1278.

```

MVC     BUFFERB(19),=CL19'/personnel/templink'
MVC     BUFLENB,=F'19'
LA      R15,BUFFERA
STG     R15,BUFA
MVC     BUFLINA,=F'1023'
SPACE  ,
CALL    BPX4RDL,              Read the value of a symbolic link +
      (BUFLENB,              Input: Linkname length          +
      BUFFERB,               Input: Link name                +
      BUFLINA,               Input: Buffer size - 1023          +
      BUFA,                  ->Buffer for symbolic link        +
      RETVAL,                Return value: 0, -1 or char count +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      MF=(E,PLIST)           -----

```

## BPX4RDV (readv) example

The following code issues a readv for a socket. SOCKDESC was returned previously from a call to either BPX4SOC or BPX4ACP. For the callable service, see [“readv \(BPX1RDV, BPX4RDV\) — Read data and store it in a set of buffers”](#) on page 673. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYIOV — Map the I/O vector structure”](#) on page 1098. AMODE 31 callers use [“BPX1RDV \(readv\) example”](#) on page 1279.

```

SPACE  ,
LA      R2,BUFFERA
STG     R2,IOV_BASE
LA      R2,L'BUFFERA
STG     R2,IOV_LEN
CALL    BPX4RDV,              Read into a vector of buffers +
      (SOCKDESC,             Input: Socket Descriptor      +
      =A(1),                 Input: Number of elements in iov +
      IOV,                   Input: Iov containing info      +
      PRIMARYALET,           Input: Alet where iov resides    +
      PRIMARYALET,           Input: Alet of buffers for data   +
      RETVAL,                Return value: 0 or -1             +
      RETCODE),              Return code                      +

```

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4RDX (read extlink) example

The following code reads the contents of external symbolic link /personnel/tmpxlink into the buffer provided. This will be the pathname that was specified when the external symbolic link was defined. For the callable service, see “[read\\_extlink \(BPX1RDX, BPX4RDX\) — Read an external symbolic link](#)” on page 663. AMODE 31 callers use “[BPX1RDX \(read extlink\) example](#)” on page 1279.

MVC	BUFFERB(19),=CL19'/personnel/tmpxlink'	
MVC	BUFLNB,=F'19'	
LA	R15,BUFFERA	
STG	R15,BUFA	
MVC	BUFLNA,=F'1023'	
SPACE	,	
CALL	BPX4RDX,	Read value of an external link +
	(BUFLNB,	Input: Linkname length +
	BUFFERB,	Input: Link name +
	BUFLNA,	Input: Buffer size - 1023 +
	BUFA,	->Buffer for symbolic link +
	RETVAL,	Return value: 0, -1 or char count +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	MF=(E,PLIST)	-----

## BPX4RD2 (readdir2) example

The following code reads multiple name entries from the specified directory (DIRECTDES). FUIOCURSOR, set to zero by the BPXYFUIO macro, indicates that the system is to begin reading with the first entry in the directory. For the callable service, see “[readdir2 \(BPX1RD2, BPX4RD2\) — Read an entry from a directory](#)” on page 659. For the data structure, see “[BPXYDIRE — Map directory entries for readdir](#)” on page 1074. AMODE 31 callers use “[BPX1RD2 \(readdir2\) example](#)” on page 1279.

MVC	DIRECTDES,..	Directory descriptor from opendir
MVC	FUIOID,=CL4'FUIO'	Eye Catcher
MVC	FUIOLEN,=AL4(FUIO#LENGTH)	length
LA	R15,BUFFERA	Set address of buffer
STG	R15,FUIOBUFF64VADDR	for directory data in FUIO
MVC	FUIOIBYTESRW,=F'1023'	Max number of bytes to read
MVI	FUIOFLAG2,FUIOADDR64	Set 64bit addressing
SPACE	,	
CALL	BPX4RD2,	Read directory entries +
	(DIRECTDES,	Input: Directory file descriptor +
	FUIO,	Input/output: BPXYFUIO +
	RETVAL,	Return value: 0, -1 or char count +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	MF=(E,PLIST)	-----

## BPX4RED (read) example

The following code reads 80 bytes from the specified file (FILEDESC) and places them in the area provided (BUFFERA). For the callable service, see “[read \(BPX1RED, BPX4RED\) — Read from a file or socket](#)” on page 652. AMODE 31 callers use “[BPX1RED \(read\) example](#)” on page 1280.

MVC	FILEDESC,..	File descriptor
LA	R15,BUFFERA	Buffer
STG	R15,BUFA	Buffer address
MVC	BUFLNA,=F'80'	Read buffer length
SPACE	,	
CALL	BPX4RED,	Read from a file +
	(FILEDESC,	Input: File descriptor +
	BUFA,	->Buffer to read into +
	PRIMARYALET,	Input: Buffer ALET +
	BUFLNA,	Input: Number of bytes to read +

RETVAL,	Return value: 0, -1, or char count+
RETCODE,	Return code
RSNCODE),	Reason code
MF=(E,PLIST)	-----

## BPX4REN (rename) example

The following code changes the directory name of a file from `usr/sam` to `usr/samantha`. For the callable service, see [“rename \(BPX1REN, BPX4REN\) — Rename a file or directory” on page 690](#). AMODE 31 callers use [“BPX1REN \(rename\) example” on page 1280](#).

MVC	BUFFERB(07),=CL07'usr/sam'	
MVC	BUFLNB,=F'07'	
MVC	BUFFERA(12),=CL12'usr/samantha'	
MVC	BUFLNA,=F'12'	
SPACE	,	
CALL	BPX4REN,	Rename a file
	(BUFLNB,	Input: Old name length
	BUFFERB,	Input: Old name
	BUFLNA,	Input: New name length
	BUFFERA,	Input: New name
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4RFM (recvfrom) example

The following code issues a `recv` from a socket. `SOCKDESC` was returned from a previous call, either `BPX4SOC` or `BPX4ACP`. For the callable service, see [“recvfrom \(BPX1RFM, BPX4RFM\) — Receive data from a socket and store it in a buffer” on page 683](#). For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants” on page 1151](#) and [“BPXYMSGF — Map the message flags” on page 1107](#). AMODE 31 callers use [“BPX1RFM \(recvfrom\) example” on page 1280](#).

SPACE	,	
MVC	MSG_FLAGS4,MSG_PEEK	
CALL	BPX4RFM,	Read from a socket
	(SOCKDESC,	Input: Socket Descriptor
	=A(L'BUFFERA),	Input: Length of the input buffer
	BUFFERA,	Input: Address of the input buffer
	PRIMARYALET,	Input: Alet of the input buffer
	MSG_FLAGS,	Input: Flags
	=A(L'SOCKADDR),	Input: Length of the socket addr
	SOCKADDR,	Input: The socket address
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4RLA (readlinkat) example

The following code creates the file for the file identified by `pathname` within the directory represented by the file descriptor `FILEFD`. Prior to this code excerpt, `FILEFD` would have been to the file descriptor of the desired directory or to `AT_FDCWD` to indicate the current directory. For the callable service, see [“readlinkat \(BPX1RLA, BPX4RLA\) — Read the value of a symbolic link relative to a directory file descriptor” on page 669](#). AMODE 31 callers use [“BPX1RLA \(readlinkat\) example” on page 1281](#).

MVC	BUFFERB(19),=CL19'/personnel/templink'	
MVC	BUFLNB,=F'19'	
LA	R15,BUFFERA	
ST	R15,BUFA	
MVC	BUFLNA,=F'1023'	
SPACE	,	
CALL	BPX4RLA,	Read the value of a symbolic link
	(BUFLNB,	Input: Linkname length
	BUFFERB,	Input: Link name
	BUFLNA,	Input: Buffer size - 1023

## BPX4RMD (rmdir) example

BUFA,	->Buffer for symbolic link	+
FILEDESC,	Input: Directory file descriptor	+
RETVAL,	Return value: 0, -1 or char count	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
VL,MF=(E,PLIST)	-----	

## BPX4RMD (rmdir) example

The following code removes directory applib/user02. For the callable service, see [“rmdir \(BPX1RMD, BPX4RMD\) — Remove a directory”](#) on page 703. AMODE 31 callers use [“BPX1RMD \(rmdir\) example”](#) on page 1281.

MVC	BUFFERA(13),=CL13'applib/user02'	
MVC	BUFLINA,=F'13'	
SPACE	,	
CALL	BPX4RMD,	Remove a directory
	(BUFLINA,	Input: Directory name length
	BUFFERA,	Input: Directory to be removed
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4RMG (resource) example

The following code retrieves system-wide resource measurement data. For the callable service, see [“resource \(BPX1RMG, BPX4RMG\) — Measure resources”](#) on page 699. For the data structure, see [“BPXYRMON — Map resource monitor data”](#) on page 1142. AMODE 31 callers use [“BPX1RMG \(resource\) example”](#) on page 1281.

CALL	BPX4RMG,	Resource measurement gatherer	+
	(RMONL,	Input: Length of BPXYRMON	+
	RMON,	Input: Buffer, BPXYRMON	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4RMS (recvmsg) example

The following code issues a recvmsg for a socket. SOCKDESC was returned from a previous call to either BPX4SOC or BPX4ACP. For the callable service, see [“recvmsg \(BPX2RMS, BPX4RMS\) — Receive messages on a socket and store them in message buffers”](#) on page 687. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151, [“BPXYMSGF — Map the message flags”](#) on page 1107, [“BPXYMSGH — Map the message header”](#) on page 1109, and [“BPXYIOV — Map the I/O vector structure”](#) on page 1098. AMODE 31 callers use [“BPX2RMS \(recvmsg\) example”](#) on page 1282.

SPACE	,		
XC	MSGH(MSGH#LENGTH),MSGH	Clear msgh	
LA	R2,SOCKADDR		
STG	R2,MSGHNAMEPTR	Store the address of sockaddr	
LA	R2,SOCK#LEN+SOCK_SUN#LEN		
ST	R2,MSGHNAMELEN		
LA	R2,IOV		
STG	R2,MSGHIOVPTR		
MVI	MSGHIOVNUM,1		
LA	R2,BUFFERA		
STG	R2,IOV_BASE		
LA	R2,L'BUFFERA		
STG	R2,IOV_LEN		
*			
CALL	BPX4RMS,	Receive a message from a socket	+
	(SOCKDESC,	Input: Socket Descriptor	+
	MSGH,	Input: Address of BPXYMSGH	+
	MSG_FLAGS,	Input: Flags	+

PRIMARYALET,	Input: Alet of the iov	+
PRIMARYALET,	Input: Alet of the buffers in iov	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4RNA (renameat) example

The following code renames a file `lnk03oldfile01.txt` relative to directory file descriptors. For the data structure, see [“BPXYFCTL — Command values and flags for fcntl”](#) on page 1076. AMODE 31 callers use the [“BPX1RNA \(renameat\) example”](#) on page 1282.

```

CALL BPX1RNA,          Rename a file          x
    (=F'18',          Length of existing file name x
    =C'lnk03oldfile01.txt', Name of exiting file x
    =F'19',          Length of new file name      x
    =C'lnk03_newfile02.txt', Name of new File name x
    Olddirfd,        Directory for old file      x
    newdirfd,        Directory for new file      x
    =A(RENAME_NOREPLACE), Flags                x
    retVal,          x
    my_retCode,      x
    rsncCode),      x
    VL,MF=(E,PLIST)

```

PLIST    DS    10A  
         BPXYFCTL

## BPX4RPH (realpath) example

The following code gets the absolute pathname without dot (`.`), dot-dot (`..`), or symbolic links for the input pathname. For the callable service, see [“realpath \(BPX1RPH, BPX4RPH\) — Resolve a path name”](#) on page 677. AMODE 31 callers use [“BPX1RPH \(realpath\) example”](#) on page 1283.

```

MVC  BUFFERA(8),=CL2'..'
MVC  BUFLINA,=F'2'
MVC  BUFLENB,=F'1024'
SPACE ,
CALL BPX4RPH,          Resolve pathname          +
    (BUFLINA,          Input: Pathname length    +
    BUFFERA,          Input: Pathname            +
    BUFLENB,          Input: Length resolved name area +
    BUFFERB,          Output: Resolved name buffer +
    RETVAL,          Return value: -1 or length  +
    RETCODE,          Return code                +
    RSNCODE),          Reason code                +
    MF=(E,PLIST)      -----

```

## BPX4RW (Pwrite) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). It will start writing at specified offset, 30 bytes from start of the file. To positional read from a file, change the FUIORWIND to indicate FUIO#RD. For the callable service, see [“pread\(\) and pwrite\(\) \(BPX1RW, BPX4RW\) — Read from or write to a file without changing the file pointer”](#) on page 572. AMODE 31 callers use [“BPX1RW \(Pwrite\) example”](#) on page 1283.

```

MVC  FILEDESC,          File descriptor from open
XC   FUIO,FUIO          Zero out Fuio fields
MVC  FUIOID,=CL4'FUIO'  Eye Catcher
MVC  FUIOLEN,=AL4(FUIO#LENGTH) length
LA   R15,BUFFERA        Set address of buffer
STG  R15,FUIOBUFFERADDR for buffer data in FUIO
MVI  FUIORWIND,FUIO#WRT Flag to indicate to PWrite

```

## BPX4RWD (rewinddir) example

MVC	FUI0IBYTESRW,=F'80'	Number of bytes to Write	
MVC	FUI0CUR2,=F'30'	Offset to start writing	
MVI	FUI0FLAG2,FUI0ADDR64	Set 64bit addressing	
LA	R15,FUI0	Set address of Fuio	
STG	R15,LFUI0PTR	For access to Fuio fields	
SPACE	,		
CALL	BPX4RW,	PWrite to a file	+
	(FILEDESC,	Input: File descriptor	+
	LFUI0PTR,	Input: Address of FUI0 struct	+
	PRIMARYALET,	Input: Fuio ALET	+
	FUI0LEN,	Input: Fuio Length	+
	RETVAL,	Return value: -1 or bytes written	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4RWD (rewinddir) example

The following code resets the open directory to the beginning. For the callable service, see [“rewinddir \(BPX1RWD, BPX4RWD\) — Reposition a directory stream to the beginning”](#) on page 701. AMODE 31 callers use [“BPX1RWD \(rewinddir\) example”](#) on page 1283.

MVC	DIRECTDES,..	File descriptor from opendir	
CALL	BPX4RWD,	Reposition directory at beginning	+
	(DIRECTDES,	Input: Directory file descriptor	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SA2 (\_\_sigactionset) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see [“\\_\\_sigactionset \(BPX1SA2, BPX4SA2\) — Examine or change a set of signal actions”](#) on page 839. For the data structure, see [“BPXYSIGH — Signal constants”](#) on page 1148. AMODE 31 callers use [“BPX1SA2 \(\\_\\_sigactionset\) example”](#) on page 1284.

XC	R15,R15		
ST	R15,SSETOPTION_FLAGS		
OI	SSETOPTION_FLAGS1,SSET_IGINVALID		
LA	R14,1		
ST	R11,BUFCNTB		
LA	R14,BUFFERA		
USING	SSET,R14		
MVC	SSETFLAGS,=XL4'00000000'		
MVC	SSETSAMASK,=XL8'0FFF0F0000000000'		
MVC	SSETSAHANDLER,EPADDR		
MVC	SSETUSERDATA,=CL4'DATA'		
DROP	R14		
SPACE	,		
CALL	BPX4SA2,	Examine/change multiple sig acts	+
	(=A(1),	Input: One SSET set	+
	BUFFERA,	Input: Signal set input BPXYSSET	+
	BUFCNTB,	In/Out: Number of array elements	+
	BUFFERB,	Output: Address of output struct	+
	SSETOPTION_FLAGS,	Input: Mapped by BPXYSSET	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SCT (semctl) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM\_ID semaphore set. For the callable service, see [“semctl \(BPX1SCT, BPX4SCT\) — Perform semaphore control](#)

operations” on page 713. For the data structure, see “BPXYSEM — Map interprocess communication semaphores” on page 1146. AMODE 31 callers use “BPX1SCT (semctl) example” on page 1284.

```

LA      R15,BUFFERA
STG     R15,BUFA
MVC     SEM_NUMBER(4),4      Semaphore number 4 in set
SPACE   ,
CALL    BPX4SCT,             Semaphore control operations      +
      (SEM_ID,               Input: Semaphore set ID          +
      SEM_NUMBER,            Input: Semaphore number (0 based) +
      =A(SEM_GETPID),         Input: Action to take          BPXYSEM +
      BUFA,                   Input: Value | Buffer | Array | 0  +
      RETVAL,                 Return value: 0, -1 or value      +
      RETCODE,                Return code                      +
      RSNCODE),               Reason code                      +
      MF=(E,PLIST)           -----

```

## BPX4SDD (setdubdefault) example

The following code sets the dub default setting for the subtasks of the caller to process. For the callable service, see “set\_dub\_default (BPX1SDD, BPX4SDD) — Set the dub default service” on page 752. AMODE 31 callers use “BPX1SDD (setdubdefault) example” on page 1284.

```

CALL    BPX4SDD,             Set effective group ID          +
      (=A(DUBPROCESS),        Input: Set Dub Constant      BPXYCONS +
      RETVAL,                  Return value: 0 or -1          +
      RETCODE,                 Return code                    +
      RSNCODE),               Reason code                    +
      MF=(E,PLIST)           -----

```

## BPX4SEC (\_\_login, \_\_login\_\_applid, \_\_certificate) example

The following code will invoke RACF (or other security product) to create a security environment (ACEE) for the calling process with the identity of JOEUSER. For the callable service, see “\_\_login, \_\_login\_\_applid, \_\_certificate (BPX1SEC, BPX4SEC) — Provides an interface to the security product” on page 364. AMODE 31 callers use “BPX1SEC (\_\_login, \_\_login\_\_applid, \_\_certificate) example” on page 1285.

```

MVC     USERNLEN,=F'7'
MVC     USERNAME(7),=CL7'JOEUSER'
MVC     OLDPASSLEN,=F'8'
MVC     OLDPASS,=CL8'JOESPASS'
MVC     OPTIONS,=F'0'
SPACE   ,
CALL    BPX4SEC,             Create security environment      +
      (=A(SEcurity.CREATE#),  Input: Function_code          BPXYCONS +
      SECURITY_USERID#,        Input: ID-Type              BPXYCONS +
      USERNLEN,                Input: UserID Length         +
      USERNAME,                Input: UserID               +
      OLDPASSLEN,              Input: Password Length       +
      OLDPASS,                 Input: Password             +
      =A(0),                   Input: Holder               +
      =A(0),                   Input: Holder               +
      OPTIONS,                 Input: Options               +
      RETVAL,                  Return value: 0 or -1         +
      RETCODE,                 Return code                   +
      RSNCODE),               Reason code                   +
      MF=(E,PLIST)           -----

```

## BPX4SEG (setegid) example

The following code sets the effective group ID of the invoker to 1. For the callable service, see “setegid (BPX1SEG, BPX4SEG) — Set the effective group ID” on page 758. AMODE 31 callers use “BPX1SEG (setegid) example” on page 1285.

```

MVC  GROUPID,=XL4'00000001' Value of new effective ID
SPACE ,
CALL  BPX4SEG,          Set effective group ID          +
      (GROUPID,         Input: Group ID                +
      RETVAL,           Return value: 0 or -1           +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4SEL (select) example

The following code issues a select for a previously connected socket. SOCKDESC was returned when the socket was created. In this case, the select is for a single socket for read, write and exception. Do not request waiting. There are no ECBs. For the callable service, see “select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues” on page 706. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151 and “BPXYSEL — Map the select options” on page 1145. AMODE 31 callers use “BPX1SEL (select) example” on page 1285.

```

*      SPACE ,
*
*      MVC  SELLIST(4),=XL4'81000000'                +
*                                           Turn on the bit representing sd 0 +
*                                           and sd 7
*
*      LA    R8,8                                     One more than largest descriptor
*      ST    R8,SOCKDESC                             Set number of sockets to check
*
*      CALL  BPX4SEL,          Select on a set of sockets          +
*      (SOCKDESC,             Input: Number of file descriptors +
*      =A(4),                 Input: Length of read list         +
*      SELLIST,               Input: Read list                   +
*      =A(4),                 Input: Length of write list         +
*      SELLIST,               Input: Write list                   +
*      =A(4),                 Input: Length of exception list     +
*      SELLIST,               Input: Exception list               +
*      =AD(0),                Input: Address of Timeout value     +
*      =AD(0),                Input: ECB pointer                  +
*      =A(SEL#BITSFORWARD),   Input: Option - bits forward       +
*      RETVAL,                Return value: 0 or -1               +
*      RETCODE,               Return code                         +
*      RSNCODE),              Reason code                         +
*      MF=(E,PLIST)          -----

```

## BPX4SEU (seteuid) example

The following code sets the effective user ID of the invoker to 1. For the callable service, see “seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID” on page 760. AMODE 31 callers use “BPX1SEU (seteuid) example” on page 1286.

```

MVC  USERID,=XL4'00000001' Value of new effective user ID
SPACE ,
CALL  BPX4SEU,          Set effective user ID          +
      (USERID,          Input: User ID                +
      RETVAL,           Return value: 0 or -1           +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```



## BPX4SF (send\_file) example

The following code create a parameter list to send the contents of the specified file to the designated socket. to 1. For the callable service, see “[send\\_file \(BPX1SF, BPX4SF\) — Send a file on a socket](#)” on page 730. AMODE 31 callers use “[BPX1SF \(send\\_file\) example](#)” on page 1286.

```

LA      R5,BUFFERA
ST      R5,BUFFR
USING   SFPL,R5
XC      SFPL(SFPL#LENGTH),SFPL Initialize to nulls (required)
* NULLS= no header, no trailer, start at offset 0
*
* MVC    SFFileDes,...           Read from file
* MVC    SFSocketDes,...        Write to Socket
MVC     SFFileBytesH,=XL4'FFFFFFFF' To file end
MVC     SFFileBytesL,=XL4'FFFFFFFF' To file end
OI      SFflagByte4,SF_Close    Close socket after write
SPACE   ,
CALL    BPX4SF,                 Send_file                +
      (=A(SFPL#LENGTH),        Input: Length of BPXYSFPL  +
      BUFFERA,                  Input: ->SFPL              +
      RETVAL,                   Return value: 0 or -1       +
      RETCODE,                  Return code                 +
      RSNCODE),                 Reason code                 +
      MF=(E,PLIST)              -----

```

## BPX4SGE (setgrent) example

The following code resets the group database to the beginning, so that a subsequent BPX4GGE call will restart the group database search from the first entry. For the callable service, see “[setgrent \(BPX1SGE, BPX4SGE\) — Reset the group database](#)” on page 765. AMODE 31 callers use “[BPX1SGE \(setgrent\) example](#)” on page 1286.

```

CALL    BPX4SGE,                 Reset the group database  +
      (RETVAL),                  Return value: 0           +
      MF=(E,PLIST)              -----

```

## BPX4SGI (setgid) example

The following code sets the real, effective, and save group IDs to 1. For the callable service, see “[setgid \(BPX1SGI, BPX4SGI\) — Set the group ID](#)” on page 762. AMODE 31 callers use “[BPX1SGI \(setgid\) example](#)” on page 1287.

```

MVC     USERID,=XL4'00000001' Value of new group user ID
SPACE   ,
CALL    BPX4SGI,                 Set group ID                +
      (GROUPID,                  Input: Group ID              +
      RETVAL,                    Return value: 0 or -1         +
      RETCODE,                   Return code                   +
      RSNCODE),                  Reason code                   +
      MF=(E,PLIST)              -----

```

## BPX4SGQ (sigqueue) example

The following code queues a signal (SIGUSR1#) to the process specified by PROCID with a signal value of 0. For the callable service, see “[sigqueue \(BPX1SGQ, BPX4SGQ\) — Queue a signal to a process](#)” on page 848. AMODE 31 callers use “[BPX1SGQ \(sigqueue\) example](#)” on page 1287.

```

SPACE   ,
CALL    BPX4SGQ,                 Queue a signal to a process  +
      (PROCID,                   Input: Process ID            +
      =A(SIGUSR1#),              Input: Signal              BPXYSIGH +
      =AD(0),                    Input: Signal value         +
      =A(0),                     Input: Signal options        +

```

## BPX4SGR (setgroups) example

RETVAL,	Return value: -1 or 0	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4SGR (setgroups) example

The following code sets the supplementary group id list to the three gids (00000001, 00000002, 00000003) in BUFFERA. For the callable service, see “[setgroups \(BPX1SGR, BPX4SGR\) — Set the supplementary group IDs list](#)” on page 766. AMODE 31 callers use “[BPX1SGR \(setgroups\) example](#)” on page 1287.

LA	R15,BUFFERA	
STG	R15,BUFA	
MVC	BUFFERA(12),=XL12'000000010000000200000003'	
SPACE	,	
CALL	BPX4SGR,	Set supplementary groups list
	(=A(3),	Input: number of sgids in list
	BUFA,	Input: address of sgids list
	RETVAL,	Return value: -1 or 0
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4SGT (semget) example

The following code creates a private set of 10 semaphores. For the callable service, see “[semget \(BPX1SGT, BPX4SGT\) — Create or find a set of semaphores](#)” on page 718. For the data structure, see “[BPXYSEM — Map interprocess communication semaphores](#)” on page 1146. AMODE 31 callers use “[BPX1SGT \(semget\) example](#)” on page 1287.

MVC	KEY(4),=A(IPC_PRIVATE)	Local to this family
MVI	S_TYPE,IPC_CREAT+IPC_EXCL	Must not already exist
MVI	S_MODE1,0	Not used
MVI	S_MODE2,S_IRUSR	All read and write permissions
MVI	S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH	
MVC	NUMB_SEMS(4),=A(10)	10 semaphores this set
SPACE	,	
CALL	BPX4SGT,	Create a set of semaphores
	(KEY,	Input: Semaphore key
	NUMB_SEMS,	Input: Number semaphores in set
	S_MODE,	Input: Flags BPXYMODE / BPXYIPC+
	RETVAL,	Return value: -1 or Semaphore ID
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----
SPACE	,	
ICM	R15,B'1111',RETVAL	Test return value
BNP	PSEUDO	Branch on semget failure
ST	R15,SEM_ID	Store SEM_ID associated with key

## BPX4SHT (shutdown) example

The following code issues a shutdown to stop socket writes to this socket connection. SOCKDESC was returned from a previous call to BPX4SOC. For the callable service, see “[shutdown \(BPX1SHT, BPX4SHT\) — Shut down all or part of a duplex socket connection](#)” on page 832. AMODE 31 callers use “[BPX1SHT \(shutdown\) example](#)” on page 1288.

SPACE	,	
CALL	BPX4SHT,	Shutdown communication
	(SOCKDESC,	Input: Socket Descriptor
	SOCK#SHUTDOWNWRITE,	Input: How - shutdown writes
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4SIA (sigaction) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see “sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action” on page 834. For the data structure, see “BPXYSIGH — Signal constants” on page 1148. AMODE 31 callers use “BPX1SIA (sigaction) example” on page 1288.

XC	NEWMASK,NEWMASK	Don't block additional signals	
LA	R15,NCATCHER	New catcher (NCATCHER=0,1 ->)	
STG	R15,NEWHANDL		
LA	R15,OCATCHER	Old catcher (NCATCHER=0,1 ->)	
STG	R15,OLDHANDL		
SPACE	,		
CALL	BPX4SIA,	Examine or change signal action	+
	(=A(SIGALRM#),	Input: Signal constant BPXYSIGH	+
	NEWHANDL,	Input: 0, ->0, ->1 or ->catcher	+
	NEWMASK,	Input: 64Bit mask of signals	+
	=A(0),	Input: Action, BPXYSIGH	+
	OLDHANDL,	0, ->XL4 (return 0, 1 ->catcher)	+
	OLDMASK,	64 bit mask of signals	+
	OLDFLAGS,	Action, BPXYSIGH	+
	=AD(0),	Data passed to signal routine	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SIN (server\_init) example

The following code connects a server address space to WLM as a server manager for the WEB subsystem type, WEB1 subsystem name, and IMWHTTP application environment. For the callable service, see “server\_init (BPX1SIN, BPX4SIN) — Server initialization” on page 742. AMODE 31 callers use “BPX1SIN (server\_init) example” on page 1288.

MVC	SUBSYSTYPE,=CL4'WEB '	WEB Subsystem Type	
MVC	SUBSYSNAME,=CL8'WEB1 '	WEB1 Subsystem Name	
MVC	APPLENV,=CL8'IMWHTTP '	IMWHTTP Application Environment	
LA	R15,=F'7'	R15 = 7	
ST	R15,PARALLELEU	7 Parallel Execution Units	
SPACE	,		
CALL	BPX4SIN,	Server_init	+
	(=A(SRV_SERVERMGR),	Input: Manager Type (Server Mgr)	+
	SUBSYSTYPE,	Input: Subsystem Type	+
	SUBSYSNAME,	Input: Subsystem Type	+
	APPLENV,	Input: Application Environment	+
	PARALLELEU,	Input: Parallel Eu	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX4SIP (sigpending) example

The following code retrieves the mask used for pending and blocked signals. For the callable service, see “sigpending (BPX1SIP, BPX4SIP) — Examine pending signals” on page 843. AMODE 31 callers use “BPX1SIP (sigpending) example” on page 1289.

CALL	BPX4SIP,	Determine pending signals	+
	(SIGRET,	Signal mask return area (XL8)	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+

## BPX4SLK (shmem\_lock) example

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4SLK (shmem\_lock) example

The following code initializes a shared memory resident lock. For the callable service, see “[shmem\\_lock \(BPX1SLK, BPX4SLK\) — Shared memory lock service](#)” on page 814. AMODE 31 callers use “[BPX1SLK \(shmem\\_lock\) example](#)” on page 1289.

XR	R15,R15	R15 = 0	
STG	R15,LOCKATTRADDR	No lock attribute Data	
SPACE	,		
CALL	BPX4SLK,	shmem_lock	+
	(=A(SLK_INIT),	INPUT: Function Code (Init)	+
	=A(SLK_NORMAL),	INPUT: Request Type (Normal)	+
	=A(SLK_SHARED),	INPUT: Lock Type (Shared)	+
	LOCKADDR,	INPUT: ->user lockword (shared mem+)	
	LOCKATTRADDR,	INPUT: Address of lock attr area	+
	LOCKTOKENADDR,	INPUT: Address of Lock Token	+
	RETVAL,	Return value: >=0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX4SLP (sleep) example

The following code suspends running for 8 seconds or until a signal is delivered (whichever comes first). For the callable service, see “[sleep \(BPX1SLP, BPX4SLP\) — Suspend execution of a process for an interval of time](#)” on page 859. AMODE 31 callers use “[BPX1SLP \(sleep\) example](#)” on page 1289.

MVC	SECONDS,=F'8'	8 seconds	
SPACE	,		
CALL	BPX4SLP,	Temporarily suspend execution	+
	(SECONDS,	Input: Sleep interval in seconds	+
	RETVAL),	Return value: 0 or sleep time	+
	MF=(E,PLIST)	-----	

## BPX4SMF (smf\_record) example

The following code tests whether SMF recording is active for a specified SMF record type, and if it is, writes an SMF record. For the callable service, see “[smf\\_record \(BPX1SMF, BPX4SMF\) — Write an SMF record](#)” on page 861. AMODE 31 callers use “[BPX1SMF \(smf\\_record\) example](#)” on page 1290.

MVC	SMF_TYPE,=F'108'	Set SMF record type	
MVC	SMF_SUBTYPE,=F'0'	Set SMF record subtype	
MVC	BUFLINA,=F'0'	Set SMF record length	
MVC	BUFA,=FD'0'	Zero SMF record address	
CALL	BPX4SMF,	smf_record	+
	(SMF_TYPE,	SMF record type	+
	SMF_SUBTYPE,	SMF record subtype	+
	BUFLINA,	SMF record length	+
	BUFA,	SMF record address set to zero	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
ICM	R15,B'1111',RETVAL	Test return value	
BNZ	QUIT	Not recording or error, quit	
SPACE	,		
MVI	BUFFERA,C' '		
MVC	BUFFERA+1(255),BUFFERA	Clear SMF record	
MVI	BUFFERA+1,100	Set length in SMF header	
MVI	BUFFERA+5,108	Set SMF type in SMF header	
MVC	BUFFERA+18(16),=CL16'Here is the data'	Set SMF record	

```

MVC SMF_TYPE,=F'108'      Set SMF record type
MVC SMF_SUBTYPE,=F'0'      Set SMF record subtype
MVC BUFLINA,=F'100'        Set SMF record length
LA  R15,BUFFERA
STG R15,BUFA               Set SMF record address
CALL BPX4SMF,              smf_record +
    (SMF_TYPE,             SMF_record type +
     SMF_SUBTYPE,          SMF record subtype +
     BUFLINA,              SMF record length +
     BUFA,                 SMF record address +
     RETVAL,               Return value: 0 or -1 +
     RETCODE,              Return code +
     RSNCODE),             Reason code +
    MF=(E,PLIST)           -----
QUIT EQU *
```

## BPX4SMS (sendmsg) example

The following code sends a message on a socket. SOCKDESC was returned from a previous call to BPX4SOC. For the callable service, see [“sendmsg \(BPX2SMS, BPX4SMS\) — Send messages on a socket ”](#) on page 734. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151, [“BPXYIOV — Map the I/O vector structure”](#) on page 1098, and [“BPXYMSGH — Map the message header”](#) on page 1109. AMODE 31 callers use [“BPX2SMS \(sendmsg\) example”](#) on page 1290.

```

XC MSGH(MSGH#LENGTH),MSGH  Clear msgh
LA  R2,SOCKADDR
STG R2,MSGHNAMEPTR         Store the address of sockaddr
LA  R2,SOCK#LEN+SOCK_SUN#LEN
ST  R2,MSGHNAMELEN
LA  R2,IOV
STG R2,MSGHIOVPTR
MVI MSGHIOVNUM,1
*
LA  R2,BUFFERA
STG R2,IOV_BASE
LA  R2,16
STG R2,IOV_LEN
MVC BUFFERA(16),=CL16'Here is the data'
*
CALL BPX4SMS,              Send a message on a socket +
    (SOCKDESC,             Input: Socket Descriptor +
     MSGH,                  Input: Address of BPXYMSGH +
     MSG_FLAGS,             Input: Flags +
     PRIMARYALET,           Input: Alet of the iov +
     PRIMARYALET,           Input: Alet of the buffers in iov +
     RETVAL,                Return value: 0 or -1 +
     RETCODE,               Return code +
     RSNCODE),             Reason code +
    MF=(E,PLIST)           -----
```

## BPX4SND (send) example

The following code issues a send for a socket. SOCKDESC was returned previously from a call to BPX4SOC. For the callable service, see [“send \(BPX1SND, BPX4SND\) — Send data on a socket ”](#) on page 726. For the data structures, see [“BPXYSOCK — Map SOCKADDR structure and constants”](#) on page 1151 and [“BPXYMSGF — Map the message flags”](#) on page 1107. AMODE 31 callers use [“BPX1SND \(send\) example”](#) on page 1291.

```

MVC BUFLINA,=F'16'
MVC BUFFERA(16),=CL16'Here is the data'
SPACE ,
CALL BPX4SND,              Send data on a socket +
    (SOCKDESC,             Input: Socket Descriptor +
     =A(L'BUFFERA),         Input: Length of input buffer +
     BUFFERA,               Input: input buffer +
     PRIMARYALET,           Input: Alet of input buffer +
     MSG_FLAGS,             Input: Flags +
     RETVAL,                Return value: 0 or -1 +
     RETCODE,               Return code +
     MF=(E,PLIST)           -----
```

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4SNS (setns) example

The following code disassociates the current process from its IPC namespace and associates it with an existing IPC namespace referred to by the input file descriptor. In this example, the target file descriptor (TARGFD) would have been returned on the open of the IPC namespace link (/proc/pid/ns/ipc). For the callable service, see “setns (BPX1SNS, BPX4SNS) — Reassociate thread with a namespace” on page 771. For the flag definitions, see “BPXYCLNP— Map clone syscall parameters” on page 1062. For the AMODE 31 example, see “BPX1SNS (setns) example” on page 1291.

CALL	BPX4SNS,	+
	(TARGFD,	+
	=A(CLONE_NEWIPC),	+
	RETVAL,	+
	RETCODE,	+
	RSNCODE),	+
	MF=(E,PLIST)	
L	R15,RETVAL	Load return value
C	R15,=F'-1'	Test for -1 return
BE	PSEUDO	Branch on error
...		

## BPX4SOC (socket or socketpair) example

The following code creates a pair of stream sockets in the AF\_UNIX domain. For the callable service, see “socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets” on page 865. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1SOC (socket or socketpair) example” on page 1291.

CALL	BPX4SOC,	Create a socket pair	+
	(=A(AF_UNIX),	Input: Domain of AF_UNIX	+
	=A(SOCK#_STREAM),	Input: Type of socket stream	+
	=A(0),	Input: Protocol of 0	+
	=A(2),	Input: Dimension of 2 for pair	+
	SOCKETS,	Input: Socket vector for return	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SOP (semop) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM\_ID semaphore set. For the callable service, see “semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations” on page 723. For the data structure, see “BPXYSEM — Map interprocess communication semaphores” on page 1146. AMODE 31 callers use “BPX1SOP (semop) example” on page 1292.

LA	R5,BUFFERA	->Utitliy buffer
STG	R5,BUFA	
USING	SEM_BUF_ELE,R5	->1st SEM_BUF_ELE
MVC	SEM_NUM(2),=AL2(0)	Semaphore number 0
MVC	SEM_OP(2),=AL2(-1)	take the resource
MVC	SEM_FLG(2),=AL2(SEM_UNDO)	flags (undo,wait)
LA	R5,SEM#BUFLN(,R5)	->next SEM_BUF_ELE
MVC	SEM_NUM(2),=AL2(2)	number 2
MVC	SEM_OP(2),=AL2(1)	release the resource
MVC	SEM_FLG(2),=AL2(IPC_NOWAIT)	flags (nowait)
LA	R5,SEM#BUFLN(,R5)	->next SEM_BUF_ELE
MVC	SEM_NUM(2),=AL2(8)	number 8
MVC	SEM_OP(2),=AL2(0)	test for no resource
MVC	SEM_FLG(2),=AL2(0)	flags (wait)

```

SPACE ,
MVC   NUMB_SEM_OPS(4),=AL2(3) number of SEM_BUF_ELE in BUFFERA
SPACE ,
CALL  BPX4SOP,           Semaphore control operations      +
      (SEM_ID,           Input: Semaphore set ID           +
      BUFA,              Input: ->SEM_BUF_ELE             BPXYSEM +
      NUMB_SEM_OPS,      Input: Action to take             +
      RETVAL,            Return value: 0, -1 or value       +
      RETCODE,           Return code                       +
      RSNCODE),          Reason code                       +
      MF=(E,PLIST)      -----

```

## BPX4SPB (queue\_interrupt) example

The following code uses the queue\_interrupt to return the last signal delivered to the signal interface routine (SIR). For the callable service, see [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered”](#) on page 648. AMODE 31 callers use [“BPX1SPB \(queue\\_interrupt\) example”](#) on page 1292.

```

CALL  BPX4SPB,           Queue the signal                  +
      (RETVAl,           Return value: 0 or -1             +
      RETCODE,          Return code                       +
      RSNCODE),         Reason code                       +
      MF=(E,PLIST)      -----

```

## BPX4SPE (setpwent) example

The following code resets the user database to the beginning, so that a subsequent BPX4GPE call will restart the user database search from the first entry. For the callable service, see [“setpwent \(BPX1SPE, BPX4SPE\) — Reset the user database ”](#) on page 782. AMODE 31 callers use [“BPX1SPE \(setpwent\) example”](#) on page 1292.

```

CALL  BPX4SPE,           Reset the user database          +
      (RETVAl,           Return value: 0                   +
      MF=(E,PLIST)      -----

```

## BPX4SPG (setpgid) example

The following code places the invoking process in its own process group (zeros indicate that the process group ID is to be set to the process ID). For the callable service, see [“setpgid \(BPX1SPG, BPX4SPG\) — Set a process group ID for job control ”](#) on page 777. AMODE 31 callers use [“BPX1SPG \(setpgid\) example”](#) on page 1293.

```

MVC   PROCID,=A(0)       Process ID - current to leader
MVC   GROUP,=A(0)        Group ID - current to leader
SPACE ,
CALL  BPX4SPG,           Set process group ID for Job Ctl +
      (PROCID,           Input: Process to be placed in grp+
      GROUP,             Input: Target group               +
      RETVAL,            Return value: 0 or -1             +
      RETCODE,           Return code                       +
      RSNCODE),          Reason code                       +
      MF=(E,PLIST)      -----

```

## BPX4SPM (sigprocmask) example

The following code changes the signal mask to block signals 1 through 16. For the callable service, see [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ”](#) on page 845. For

## BPX4SPN (spawn) example

the data structure, see “BPXYSIGH — Signal constants” on page 1148. AMODE 31 callers use “BPX1SPM (sigprocmask) example” on page 1293.

```
LA    R15,=XL8'FFFF000000000000'  Block signals 1 thru 16
ST    R15,NEWMASKA                  New mask address
LA    R15,OLDMASK                   Old signal mask
ST    R15,OLDMASKA                  Old mask address
SPACE ,
CALL  BPX4SPN,                      Examine or change signal mask  +
      (=A(SIG_BLOCK#),              Input: How parameter BPXYSIGH  +
      NEWMASKA,                      Input: 0, ->CL8              +
      OLDMASKA,                      Input: 0 | ->returned mask  +
      RETVAL,                        Return value: 0 or -1        +
      RETCODE,                       Return code                  +
      RSNCODE),                      Reason code                  +
      MF=(E,PLIST)                   -----
```

## BPX4SPN (spawn) example

The program ictasma located at ict/bin gets control as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. The file descriptor count is set to 0, indicating that the child shall inherit all of the parent's file descriptors. The inheritance area passed is set to all zeroes, indicating that the child shall inherit the parent's attributes without change. For the callable service, see “spawn (BPX1SPN, BPX4SPN) — Spawn a process” on page 869. AMODE 31 callers use “BPX1SPN (spawn) example” on page 1293.

```

MVC   BUFLINA,=F'16'
MVC   BUFFERA(16),=C'/ict/bin/ictasma'
MVC   ARGCNT,=F'3'
*
LA     R15,=F'4'                     First
ST     R15,ARGLLST+00                Length
LA     R15,=CL4'WK18'                Length parm list
STG    R15,ARGSLST+00                Argument
*                                     Argument address parm list
LA     R15,=F'7'                     Second
ST     R15,ARGLLST+04                Length
LA     R15,=CL7'DEPT37A'              Length parm list
STG    R15,ARGSLST+08                Argument
*                                     Argument address parm list
LA     R15,=F'22'                    Third
ST     R15,ARGLLST+08                Length
LA     R15,=CL22'RATE(STD,NOEXC,NOSPEC)' Length parm list
STG    R15,ARGSLST+16                Argument
*                                     Argument address parm list
MVC   ENVCNT,=F'0'                   Zero environment args passed
MVC   ENVLENS,=F'0'                  Addr of env. data length list
MVC   ENVPARMS,=F'0'                 Add of env. data
*
MVC   FDCNT,=F'0'                    Zero file descriptors passed
MVC   FDLST,=F'0'                   File Descriptor list
*
XC     INHE(INHE#LENGTH),INHE        Clear Inheritance structure
SPACE ,
CALL  BPX4SPN,                        +
      (BUFLINA,                      Input: Pathname length  +
      BUFFERA,                       Input: Pathname        +
      ARGCNT,                        Input: Argument count  +
      ARGLLST,                       Input: Argument length list +
      ARGSLST,                       Input: Argument address list +
      ENVCNT,                        Input: Environment count  +
      ENVLENS,                       Input: Environment length list +
      ENVPARMS,                      Input: Environment address list +
      FDCNT,                         Input: File desriptor count +
      FDLST,                         Input: File descriptor list +
      =A(INHE#LENGTH),               Input: Length of Inheritance area +
      INHE,                          Input: Inheritance area  +
      RETVAL,                        Return value: Child PID or -1 +
      RETCODE,                       Return code            +
      RSNCODE),                      Reason code            +
      MF=(E,PLIST)                   -----
```



## BPX4SPR (setpeer) example

The following code issues a setpeer to set up the host address. For the callable service, see “setpeer (BPX1SPR, BPX4SPR) — Preset the peer address associated with a socket ” on page 775. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151. AMODE 31 callers use “BPX1SPR (setpeer) example” on page 1294.

```
CALL BPX4SPR,          Select on a set of sockets      +
   (SOCKDESC,          Input: Socket Descriptor        +
    SOCK#LEN+SOCK_SUN#LEN, Input: Length of socket address +
    SOCKADDR,          Input: Socket address            +
    SOCK#SO_SET,       Input: Option - set the address    +
    RETVAL,            Return value: 0 or -1             +
    RETCODE,           Return code                      +
    RSNCODE),          Reason code                     +
    MF=(E,PLIST)       -----
```

## BPX4SPW (server\_pwu) example

The following code puts work to the WLM work queue for the IMWHTTP application environment for transaction class A. For the callable service, see “server\_pwu (BPX1SPW, BPX4SPW) — Server process work unit ” on page 746. AMODE 31 callers use “BPX1SPW (server\_pwu) example” on page 1294.

```
MVC  APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
MVC  TRXCLASS,=CL8'A      ' Transaction Class A
XR   R15,R15              R15 = 0
ST   R15,CLASSIFYLEN      No Classification Data
ST   R15,APPLDATALEN      No Application Data
ST   R15,FDLISTPTR        No File Descriptor List
SPACE ,
CALL BPX4SPW,             Server_pwu                  +
   (=A(SRV_PUT_NEWWRK),  Input: Function Code (Putwork)  +
    TRXCLASS,            Input: Transaction Class        +
    APPLENV,             Input: Application Environment   +
    CLASSIFYLEN,         Input: Classification Area Length +
    CLASSIFYAREAPTR,     Input: Classification Area Address+
    APPLDATALEN,         Input: Application Data Length   +
    APPLDATAPTR,         Input: Application Data Address  +
    FDLISTPTR,           Input: Mapped by BPXYSFDL        +
    RETVAL,              Return value: 0 or -1            +
    RETCODE,             Return code                     +
    RSNCODE),            Reason code                     +
    MF=(E,PLIST)         -----
L    R15,RETVAL           Load return value
C    R15,=F'-1'           Test for -1 return
BE   PSEUDO               Branch on error
```

## BPX4SPY (setpriority) example

The following code sets the CPU priority based on the input which and who values. The which value used is PRIO\_PROCESS, which indicates that the priority is to be set by process ID. The who value used is 7, to set the priority for process ID 7. For the callable service, see “setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process” on page 779. AMODE 31 callers use “BPX1SPY (setpriority) example” on page 1295.

```
MVC  PROCID,=XL4'00000007' Process ID to set priority for
MVC  PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL BPX4SPY,             Set priority value            +
   (=A(PRIO_PROCESS),    Input: Set by Process ID        +
    PROCID,              Input: PID to set priority for    +
    PRIORITY,            Input: Priority value to set to    +
    RETVAL,              Return value: 0 or -1             +
    RETCODE,             Return code                      +
    RSNCODE),            Reason code                     +
    MF=(E,PLIST)         -----
L    R15,RETVAL           Load return value
```

C	R15,=F'-1'	Test for -1 return
BE	PSEUDO	Branch on error

## BPX4SRG (setregid) example

The following code sets the real and/or effective group IDs to 1. For the callable service, see “setregid (BPX1SRG, BPX4SRG) — Set the real and effective GIDs” on page 784. AMODE 31 callers use “BPX1SRG (setregid) example” on page 1295.

MVC	RGID,=XL4'00000001'	Value of new real group ID	
MVC	RGID,..	Group ID to be set from a getgid	
MVC	EGID,=XL4'00000001'	Value of new effective group ID	
MVC	EGID,..	Group ID to be set from a getegid	
SPACE	,		
CALL	BPX4SRG,	Set Group IDs	+
	(RGID,	Input: Real Group ID to be set	+
	EGID,	Input: Eff. Group ID to be set	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SRL (setrlimit) example

The following code sets the resource limits for the calling process based on the input resource value and the resource limits set in the input rlimit structure. The resource value is set to RLIMIT\_CPU. The resource limits are set to RLIM\_INFINITY. For the callable service, see “setrlimit (BPX1SRL, BPX4SRL) — Set resource limits” on page 789. For the data structure, see “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1142. AMODE 31 callers use “BPX1SRL (setrlimit) example” on page 1295.

MVC	RESOURCE,=A(RLIMIT_CPU)	Value of resource	
XC	RLIM_CUR_HW,RLIM_CUR_HW	Current limit highword (Zero)	
XC	RLIM_MAX_HW,RLIM_MAX_HW	Maximum limit highword (Zero)	
MVC	RLIM_CUR,=A(RLIM_INFINITY)	Current limit	
MVC	RLIM_MAX,=A(RLIM_INFINITY)	Maximum limit	
SPACE	,		
CALL	BPX4SRL,	Set resource limits	+
	(RESOURCE,	Input: resource	+
	RLIMIT,	Structure, mapped by BPXYRLIM	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
L	R15,RETVAL	Load return value	
C	R15,=F'-1'	Test for -1 return	
BE	PSEUDO	Branch on error	

## BPX4SRU (setreuid) example

The following code sets the real and/or effective user IDs to 1. For the callable service, see “setreuid (BPX1SRU, BPX4SRU) —Set the real and effective UIDs” on page 786. AMODE 31 callers use “BPX1SRU (setreuid) example” on page 1296.

MVC	RUID,=XL4'00000001'	Value of new real user ID	
MVC	RUID,..	User ID to be set from a getuid	
MVC	EUID,=XL4'00000001'	Value of new effective user ID	
MVC	EUID,..	User ID to be set from a geteuid	
SPACE	,		
CALL	BPX4SRU,	Set user IDs	+
	(RUID,	Input: Real User ID to be set	+
	EUID,	Input: Eff. User ID to be set	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SRX (srx\_np) example

srx\_np callable service sends or receives data on a socket using CSM buffers. The following example receives data into CSM buffers. The MSGXNAMEPTR is set up to point to a buffer to receive the source address of the data. The MSGXIOVX is an IVTBUFL structure, which describes an IOVX array in a CSM buffer. The IOVX array contains IVTBUFL structures, each of which describes a CSM buffer with data that was received. SOCKDESC is a socket descriptor that was returned from a previous call to either BPX4SOC or BPX4ACP. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1151 and “BPXYMSGX — Map the message header” on page 1109. For the callable service, see “srx\_np (BPX1SRX, BPX4SRX) — Send or receive CSM buffers on a socket” on page 886. AMODE 31 callers use “BPX1SRX (srx\_np) example” on page 1296.

```

XC    MSGX(MSGX#LEN),MSGX    Clear msgx storage
LA    R2,SOCKADDR
ST    R2,MSGXNAMEPTR        Store the address of sockaddr
LA    R2,SOCK#LEN+SOCK_SIN#LEN
ST    R2,MSGXNAMELEN        Length of sockaddr buffer
SPACE ,
CALL  BPX4SRX,              Receive data in CSM buffers      +
      (SOCKDESC,            Input: Socket Descriptor        +
      MSGX_RECV,            Input: Direction                +
      L'MSGX,               Input: Msghdrx length           +
      MSGX,                 Input: Msghdrx                  +
      RETVAL,               Return value: -1 or bytes read   +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          -----

```

## BPX4SSI (setsid) example

The following code creates a session and a process group (and is the leader of both). For the callable service, see “setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID” on page 793. AMODE 31 callers use “BPX1SSI (setsid) example” on page 1296.

```

CALL  BPX4SSI,              Create session, set process grp ID+
      (RETVAL,              Return value: -1 or new session ID+
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          -----

```

## BPX4SSU (sigsuspend) example

The following code replaces the invoker's current mask to block signals 1 through 16 and suspend until a signal is delivered. For the callable service, see “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 851. AMODE 31 callers use “BPX1SSU (sigsuspend) example” on page 1297.

```

MVC   WAITMASK(8),=XL8'FFFF000000000000'   Blocks 1 thru 16
SPACE ,
CALL  BPX4SSU,              Wait for a signal                +
      (WAITMASK,            Input: Wait mask, XL8            +
      RETVAL,               Return value: -1 or not returned  +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          -----

```

## BPX4STA (stat) example

The following code obtains status about file labrec/qual/current . For the callable service, see “stat (BPX1STA, BPX4STA) — Get status information about a file by path name” on page 893. For the data

## BPX4STE (set\_timer\_event) example

structure, see “BPXYSTAT — Map the response structure for stat” on page 1162. AMODE 31 callers use “BPX1STA (stat) example” on page 1297.

```
MVC  BUFFERA(19),=CL19'labrec/qual/current'
MVC  BUFLINA,=F'19'
SPACE ,
CALL  BPX4STA,          Get file status              +
      (BUFLINA,         Input: Pathname length      +
      BUFFERA,          Input: Pathname             +
      STATL,            Input: Length of buffer needed +
      STAT,             Buffer, BPXYSTAT             +
      RETVAL,           Return value: 0 or -1        +
      RETCODE,          Return code                 +
      RSNCODE),         Reason code                 +
      MF=(E,PLIST)      -----
```

## BPX4STE (set\_timer\_event) example

The following code sets a timer event, which when it expires will post the ECB represented by THLITIMERECEB. For the callable service, see “set\_timer\_event (BPX1STE, BPX4STE) — Set DIE-mode timer event” on page 798. AMODE 31 callers use “BPX1STE (set\_timer\_event) example” on page 1297.

```
CALL  BPX4STE,          Set timer event              +
      (=A(2),           Input: Number of seconds    +
      =A(500000000),    Input: Number of nanoseconds +
      RETVAL,           Return value: 0 or -1        +
      RETCODE,          Return code                 +
      RSNCODE),         Reason code                 +
      MF=(E,PLIST)      -----
```

## BPX4STF (w\_statvfs) example

The following code obtains information about file system TESTLIB.FILESYS1. For the callable service, see “w\_statvfs (BPX1STF, BPX4STF) — Get the file system status” on page 1029. For the data structure, see “BPXYSSTF — Map response structure for file system status” on page 1161. AMODE 31 callers use “BPX1STF (w\_statvfs) example” on page 1297.

```
MVC  FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX4STF,          Get file system status        +
      (FSNAME,          Input: File system name (44 char) +
      SSTFL,            Input: Length of BPXYSSTF        +
      SSTF,             Buffer, BPXYSSTF                 +
      RETVAL,           Return value: -1 or length status +
      RETCODE,          Return code                     +
      RSNCODE),         Reason code                     +
      MF=(E,PLIST)      -----
```

## BPX4STL (set\_thread\_limits) example

The following code sets the MAX\_THREAD and MAX\_THREAD\_TASKS limits for pthread\_created threads in the invoker's process. For the callable service, see “set\_thread\_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread\_created threads” on page 795. AMODE 31 callers use “BPX1STL (set\_thread\_limits) example” on page 1298.

```
CALL  BPX4STL,          Set_thread_limits            +
      (=A(STL_SET_BOTH), Input: action                +
      =A(50),            Input: new task limit        +
      =A(100),           Input: new thread limit      +
      RETVAL,            Return value: 0 or -1        +
      RETCODE,           Return code                  +
      RSNCODE),          Reason code                  +
      MF=(E,PLIST)      -----
```

## BPX4STO (sendto) example

The following code issues a sendto for a socket. SOCKDESC was returned from a previous call to either BPX4SOC or BPX4ACP. For the callable service, see “[sendto \(BPX1STO, BPX4STO\) — Send data on a socket](#)” on page 738. For the data structures, see “[BPXYSOCK — Map SOCKADDR structure and constants](#)” on page 1151 and “[BPXYMSGF — Map the message flags](#)” on page 1107. AMODE 31 callers use “[BPX1STO \(sendto\) example](#)” on page 1298.

```

MVC  BUFFERA(16),=CL16'Here is the data'
LA    R2,BUFFERA
STG   R2,I0V_BASE
MVI   I0V_LEN,16
SPACE ,
CALL  BPX4STO,          Send data to a socket          +
      (SOCKDESC,        Input: Socket Descriptor      +
      =A(L'BUFFERA),    Input: Length of the input buffer +
      BUFFERA,           Input: input buffer           +
      PRIMARYALET,      Input: Alet of the input buffer +
      MSG_FLAGS,        Input: Flags                   +
      =A(L'SOCKADDR),   Input: Length of the socket addr +
      SOCKADDR,         Input: The socket address      +
      RETVAL,           Return value: 0 or -1          +
      RETCODE,          Return code                   +
      RSNCODE),         Reason code                   +
      MF=(E,PLIST)      -----

```

## BPX4STR (setitimer) example

The following code returns the time remaining an alarm, or ITIMER\_REAL as set by setitimer. For the callable service, see “[setitimer \(BPX1STR, BPX4STR\) — Set the value of the interval timer](#)” on page 768. For the data structure, see “[BPXYITIM — Map getitimer, setitimer structure](#)” on page 1101. AMODE 31 callers use “[BPX1STR \(setitimer\) example](#)” on page 1298.

```

LA    R15,2              Initial value 2.5 seconds
ST    R15,ITIMISECONDS
L     R15,=A(500000)
ST    R15,ITIMIMICROSEC
L     R15,0              No reload value
ST    R15,ITIMRSECONDS
ST    R15,ITIMRMICROSEC
LA    R15,ITIM           Output mapping structure
STG   R15,ITIMA          ->structure
CALL  BPX4STR,          Get process data              +
      (=A(ITIMER_REAL), Input: Relative process token +
      ITIMA,             In : ->Buffer, mapped by BPXYITIM +
      ITIMA,             Out: ->Buffer, mapped by BPXYITIM +
      RETVAL,           Return value: -1, 0              +
      RETCODE,          Return code                    +
      RSNCODE),         Reason code                    +
      MF=(E,PLIST)      -----

```

## BPX4STV (statvfs) example

The following code obtains information about the file system containing the file identified by pathname. For the callable service, see “[statvfs \(BPX1STV, BPX4STV\) — Get the file system status](#)” on page 896. For the data structure, see “[BPXYSSTF — Map response structure for file system status](#)” on page 1161. AMODE 31 callers use “[BPX1STV \(statvfs\) example](#)” on page 1299.

```

MVC  BUFFERA(8),=CL8'/usr/inv'
MVC  BUFLINA,=F'8'
SPACE ,
CALL  BPX4STV,          Get file system status          +
      (BUFLINA,         Input: Pathname length          +
      BUFFERA,          Input: Pathname                 +
      SSTFL,            Input: Length of BPXYSSTF        +
      SSTF,             Buffer, BPXYSSTF                 +
      RETVAL,           Return value: -1 or length status +

```

## BPX4STW (sigtimedwait) example

RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4STW (sigtimedwait) example

The following code will wait for signals 1-4 to arrive or 3 seconds, whichever occurs first. For the callable service, see [“sigtimedwait \(BPX1STW, BPX4STW\) — Wait for a signal with a specified timeout”](#) on page 854. AMODE 31 callers use [“BPX1STW \(sigtimedwait\) example”](#) on page 1299.

MVC	WAITMASK(8),=XL8'F000000000000000'	Signals 1-4	
LA	R15,SIGINFO_T		
STG	R15,SINFA		
MVC	SECONDS,=F'3'	Wait three seconds	
XC	NANOSECONDS,NANOSECONDS	Zero nanoseconds	
SPACE	,		
CALL	BPX4STW,	Signal timed wait	+
	(WAITMASK,	Input: mask of signal to wait for	+
	SINFA,	Input: address of siginfo_t area	+
	SIGINFO#LENGTH,	Input: length of siginfo_t area	+
	SECONDS,	Input: seconds to wait for sig	+
	NANOSECONDS,	Input: nanoseconds to wait for sig	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SUI (setuid) example

The following code sets the real, effective, and saved user IDs to 1. For the callable service, see [“setuid \(BPX1SUI, BPX4SUI\) — Set user IDs”](#) on page 800. AMODE 31 callers use [“BPX1SUI \(setuid\) example”](#) on page 1299.

MVC	USERID,=XL4'00000001'	Value of new user ID	
MVC	USERID,..	User ID to be set from a getuid	
SPACE	,		
CALL	BPX4SUI,	Set user ID	+
	(USERID,	Input: User ID to be set	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SWT (sigwait) example

The following code waits for an asynchronous signal, SIGALRM bit 14 in the mask. For the callable service, see [“sigwait \(BPX1SWT, BPX4SWT\) — Wait for a signal”](#) on page 857. For the data structure, see [“BPXYSIGH — Signal constants”](#) on page 1148. AMODE 31 callers use [“BPX1SWT \(sigwait\) example”](#) on page 1300.

MVC	WAITMASK(8),=XL8'00040000000000000000'		
SPACE	,		
CALL	BPX4SWT,	Wait for asynchronous signal	+
	(WAITMASK,	Input: Signal mask SIGALRM	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4SYA (symlinkat) example

The following code creates a symbolic link relative to a directory file descriptor. For the callable service, use [“symlinkat \(BPX1SYA, BPX4SYA\) — Create a symbolic link relative to a directory file descriptor”](#) on page 905. AMODE 31 callers use the [“BPX1SYA \(symlinkat\) example”](#) on page 1300.

```

MVC FILEDESC,..          From opendir
MVC BUFFERA(12),=CL12'/sys12/acctn'
MVC BUFLINA,=F'12'
MVC BUFFERB(09),=CL09'/sysaccts'
MVC BUFLINB,=F'09'
SPACE ,
CALL BPX4SYA,             Create symbolic link to pathname +
    (BUFLINA,             Input: Old Pathname length      +
    BUFFERA,              Input: Old Pathname            +
    FILEDESC,             Input: Directory File descriptor +
    BUFLINB,              Input: New Pathname length.      +
    BUFFERB,              Input: New Pathname            +
    RETVAL,               Return value: 0 or -1           +
    RETCODE,              Return code                     +
    RSNCODE),             Reason code                     +
    VL,MF=(E,PLIST) -----

```

## BPX4SYC (sysconf) example

The following code gets the maximum number of children allowed by the configuration variable. For the callable service, see [“sysconf \(BPX1SYC, BPX4SYC\) — Determine system configuration options”](#) on page 913. For the data structure, see [“BPXYCONS — Constants used by services”](#) on page 1062. AMODE 31 callers use [“BPX1SYC \(sysconf\) example”](#) on page 1300.

```

CALL BPX4SYC,             Get configuration variable      +
    (=A(SC_CHILD_MAX),    Input: Config variable BPXYCONS +
    RETVAL,               Return value: -1 or variable    +
    RETCODE,              Return code                     +
    RSNCODE),             Reason code                     +
    MF=(E,PLIST) -----

```

## BPX4SYF (syncfs) example

The following code synchronizes the file system file referred to by the open file descriptor. For the callable service, see [“syncfs \(BPX1SYF, BPX4SYF\) — Synchronizes the file system file referred to by the open file descriptor”](#) on page 912. AMODE 31 callers use [“BPX1SYF \(syncfs\) example”](#) on page 1300.

```

CALL BPX4SYF( FileDesc,   Synfs                        +
    RETVAL,               Return value: 0 or -1          +
    RETCODE,              Return code                     +
    RSNCODE),             Reason code                     +
    VL,MF=(E,PLIST) -----

```

## BPX4SYM (symlink) example

The following code creates a symbolic link /sysaccts for path name /sys12/acctn. For the callable service, see [“symlink \(BPX1SYM, BPX4SYM\) — Create a symbolic link to a path name”](#) on page 900. AMODE 31 callers use [“BPX1SYM \(symlink\) example”](#) on page 1301.

```

MVC BUFFERA(12),=CL12'/sys12/acctn'
MVC BUFLINA,=F'12'
MVC BUFFERB(09),=CL09'/sysaccts'
MVC BUFLINB,=F'09'
SPACE ,
CALL BPX4SYM,             Create symbolic link to pathname +
    (BUFLINA,             Input: Pathname length          +
    BUFFERA,              Input: Pathname                  +
    BUFLINB,              Input: Link name length          +
    BUFFERB,              Input: Link name                 +
    RETVAL,               Return value: 0 or -1           +
    RETCODE,              Return code                     +
    RSNCODE),             Reason code                     +
    MF=(E,PLIST) -----

```

## BPX4SYN (sync) example

The following code causes all information in memory that updates file systems to be scheduled for writing out to disk. For the callable service, see “[sync \(BPX1SYN, BPX4SYN\) — Schedule file system updates](#)” on page 910. AMODE 31 callers use “[BPX1SYN \(sync\) example](#)” on page 1301.

CALL	BPX4SYN,	Sync	+
	(RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4TAF (MVSThreadAffinity) example

The following code executes the assembler routine EXITRTN on another thread, identified by thread ID THID, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see “[MVSThreadAffinity \(BPX1TAF, BPX4TAF\) — MVS thread affinity service](#)” on page 492. AMODE 31 callers use “[BPX1TAF \(MVSThreadAffinity\) example](#)” on page 1301.

*	MVC	EXITRTNA,=AD(EXITRTN)	->Routine address	
	MVC	EXITPLA,=AD(EXITPARM)	->Input parameter list	
	SPACE	,		
	CALL	BPX4TAF,		+
		(EXITRTNA,	Input: Routine address	+
		EXITPLA,	Input: Parm list address or 0	+
		THID,	Input: Target pthread to run exit	+
		RETVAL,	Return value: -1 or not return	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		MF=(E,PLIST)	-----	

## BPX4TAK (takesocket) example

The following code takes a socket that was given by the program identified by CID (clientid). SOCKDESC and CID information are passed by the program that did the givesocket (BPX4GIV). SOCKDESC is the giver's descriptor. When takesocket completes successfully, RETVAL will contain the taker's new socket descriptor. For the callable service, see “[takesocket \(BPX1TAK, BPX4TAK\) — Acquire a socket from another program](#)” on page 916. For the data structure, see “[BPXYCID — Map the returning structure for getclientid\(\)](#)” on page 1061. AMODE 31 callers use “[BPX1TAK \(takesocket\) example](#)” on page 1301.

CALL	BPX4TAK,	take a socket from another program+
	(CID,	Input: Clientid of giver
	SOCKDESC,	Input: Giver's socket descriptor
	RETVAL,	Return value: -1 or new descriptor+
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----
L	R2,RETVAL	
ST	R2,SOCKDES2	Store the new socket descriptor

## BPX4TDR (tcdrain) example

The following code waits until all output sent to the standard output file has been transmitted. For the callable service, see “[tcdrain \(BPX1TDR, BPX4TDR\) — Wait until output has been transmitted](#)” on page 918. AMODE 31 callers use “[BPX1TDR \(tcdrain\) example](#)” on page 1302.

CALL	BPX4TDR,	Wait for output transmittal	+
	(=A(STDOUT_FILENO),	Input: File descriptor	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+



RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4TFH (tcflush) example

The following code flushes all the data in the standard input file. the callable service, see “[tcflush \(BPX1TFH, BPX4TFH\) — Flush input or output on a terminal](#)” on page 923. For the data structure, see “[BPXYTIOS — Map the termios structure](#)” on page 1169. AMODE 31 callers use “[BPX1TFH \(tcflush\) example](#)” on page 1302.

CALL	BPX4TFH,	Line control flush	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	=A(TCIFLUSH),	Input: Queue selector BPXYTIOS	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4TFW (tcflow) example

The following code resumes data flow (TCION transmits a START character) on the standard input file. For the callable service, see “[tcflow \(BPX1TFW, BPX4TFW\) — Suspend or resume data flow on a terminal](#)” on page 921. For the data structure, see “[BPXYTIOS — Map the termios structure](#)” on page 1169. AMODE 31 callers use “[BPX1TFW \(tcflow\) example](#)” on page 1302.

CALL	BPX4TFW,	Suspend or resume data flow	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	=A(TCION),	Input: Action BPXYTIOS	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4TGA (tcgetattr) example

The following code retrieves control information about the standard input file. For the callable service, see “[tcgetattr \(BPX1TGA, BPX4TGA\) — Get the attributes for a terminal](#)” on page 926. For the data structure, see “[BPXYTIOS — Map the termios structure](#)” on page 1169. AMODE 31 callers use “[BPX1TGA \(tcgetattr\) example](#)” on page 1302.

CALL	BPX4TGA,	Get a terminal control structure	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	TIOS,	Termio structure, BPXYTIOS	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4TGC (tcgetcp) example

The following code retrieves information about code page change notification (CPCN) capability and the BPXYTCCP structure. For the callable service, see “[tcgetcp \(BPX1TGC, BPX4TGC\) — Get terminal code page names](#)” on page 928. For the data structure, see “[BPXYTCCP — Map the terminal control code page structure](#)” on page 1163. AMODE 31 callers use “[BPX1TGC \(tcgetcp\) example](#)” on page 1303.

CALL	BPX4TGC,	Get code page names	+
	(=A(STDIN_FILENO),	Input: File descriptor	+
	=A(TCCP#LENGTH),	Input: Length of BPXYTCCP	+
	TCCP,	Output: Termcp structure BPXYTCCP	+
	RETVAL,	Return value: 0 or -1	+
	RETCODE,	Return code	+

## BPX4TGP (tcgetpgrp) example

RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4TGP (tcgetpgrp) example

The following code gets the foreground process group ID associated with the controlling terminal. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see “[tcgetpgrp \(BPX1TGP, BPX4TGP\) – Get the foreground process group ID](#)” on page 931. AMODE 31 callers use “[BPX1TGP \(tcgetpgrp\) example](#)” on page 1303.

CALL	BPX4TGP,	Get the foreground process grp ID +
	(=A(STDIN_FILENO)),	Input: File descriptor +
	RETVAL,	Return value -1, fgpd proc grp ID +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	MF=(E,PLIST)	-----

## BPX4TGS (tcgetsid) example

The following code retrieves the process group ID of the session for which the terminal specified by file descriptor is the controlling terminal. For the callable service, see “[tcgetsid \(BPX1TGS, BPX4TGS\) – Get a process group ID for the session leader for the controlling terminal](#)” on page 933. AMODE 31 callers use “[BPX1TGS \(tcgetsid\) example](#)” on page 1303.

CALL	BPX4TGS,	Get session process group ID +
	(=A(STDIN_FILENO)),	Input: File descriptor +
	RETVAL,	Return value: 0 or -1 +
	RETCODE,	Return code +
	RSNCODE),	Reason code +
	MF=(E,PLIST)	-----

## BPX4TIM (times) example

The following code gathers selected times about the invoker's CPU utilization. For the callable service, see “[times \(BPX1TIM, BPX4TIM\) – Get process and child process times](#)” on page 950. For the data structure, see “[BPXYTIMS – Map the response structure for times](#)” on page 1168. AMODE 31 callers use “[BPX1TIM \(times\) example](#)” on page 1303.

CALL	BPX4TIM,	Process CPU times	+
	(TIMS,	Input: Buffer	BPXYTIMS +
	RETVAL,	Return value: -1 or clock_t	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	

## BPX4TLS (pthread\_security\_np) example

The following code creates a thread-level security environment for the calling thread using the identity specified by the caller. For the callable service, see “[pthread\\_security\\_np, pthread\\_security\\_applid\\_np \(BPX1TLS, BPX4TLS\) – Create or delete thread-level security](#)” on page 601. For the data structure, see “[BPXYCONS – Constants used by services](#)” on page 1062. AMODE 31 callers use “[BPX1TLS \(pthread\\_security\\_np\) example](#)” on page 1304.

MVC	IDENT,=CL8'USERID05'		
MVC	PASSWORD,=CL7'MYPSWRD'		
SPACE	,		
CALL	BPX4TLS,	pthread_security_np	+
	(=A(TLS_CREATE_THREAD_SEC#),	Input: Func_code	BPXYCONS +
	TLS_IDENTITY_USERID#,	Input: Identity_type	BPXYCONS +
	=A(8),	Input: Identity length	+
	IDENT,	Input: Identity	+

=A(7),	Input: Password length	+
PASSWORD,	Input: Password	+
RETVAL,	Return value: 0 or -1	+
RETCODE,	Return code	+
RSNCODE),	Reason code	+
MF=(E,PLIST)	-----	

## BPX4TRU (truncate) example

The following code truncates the file described by /somedir/somefile.c to a length of 512 bytes. For the callable service, see [“truncate \(BPX1TRU, BPX4TRU\) — Change the size of a file ” on page 953](#). AMODE 31 callers use [“BPX1TRU \(truncate\) example” on page 1304](#).

MVC	BUFFERA(20),=CL20'/somedir/somefile.c'	
MVC	BUFLINA,=F'20'	
MVC	NEWLEN(8),=FL8'512'	
SPACE	,	
CALL	BPX4TRU,	Truncate a file
	(BUFLINA,	Input: Pathname length
	BUFFERA,	Input: Pathname
	NEWLEN,	Input: Length to keep
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4TSA (tcsetattr) example

The following code turns off the HUPCL (hang up on last close) bit for the standard input file. For the callable service, see [“tcsetattr \(BPX1TSA, BPX4TSA\) — Set the attributes for a terminal ” on page 937](#). For the data structure, see [“BPXYTIOS — Map the termios structure” on page 1169](#). AMODE 31 callers use [“BPX1TSA \(tcsetattr\) example” on page 1304](#).

NI	C_CFLAG+HUPCL_0,X'FF'-HUPCL	Turn off HUPCL
*	termios was retrieved by a prior tcgetattr	
CALL	BPX4TSA,	Set terminal attributes
	(=A(STDIN_FILENO),	Input: File descriptor
	=A(TCSADRAIN),	Input: Action
	TIOS,	Input: Terminos struct
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4TSB (tcsendbreak) example

The following code requests that a break be sent to the standard input file. For the callable service, see [“tcsendbreak \(BPX1TSB, BPX4TSB\) — Send a break condition to a terminal ” on page 934](#). AMODE 31 callers use [“BPX1TSB \(tcsendbreak\) example” on page 1305](#).

CALL	BPX4TSB,	Send break condition to terminal
	(=A(STDIN_FILENO),	Input: File descriptor
	=A(0),	Duration, not used in z/OS UNIX
	RETVAL,	Return value: 0 or -1
	RETCODE,	Return code
	RSNCODE),	Reason code
	MF=(E,PLIST)	-----

## BPX4TSC (tcsetcp) example

The following code sets code page names and Code Page Change Notification (CPCN) capability. For the callable service, see [“tcsetcp \(BPX1TSC, BPX4TSC\) — Set terminal code page names ” on page 940](#).

## BPX4TSP (tcsetpgrp) example

For the data structure, see [“BPXYTCCP — Map the terminal control code page structure”](#) on page 1163. AMODE 31 callers use [“BPX1TSC \(tcsetcp\) example”](#) on page 1305.

```
XC      TCCP(TCCP#LENGTH),TCCP  Clear area
OI      TCCPFLAGB4,TCCPFASTP  Set local translation
MVC     TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
MVC     TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
SPACE   ,
CALL    BPX4TSP,                Set code page names          +
      (=A(STDIN_FILENO),        Input: File descriptor      +
      =A(TCCP#LENGTH),          Input: Length of BPXYTCCP    +
      TCCP,                     Termcp structure, BPXYTCCP    +
      RETVAL,                   Return value: 0 or -1         +
      RETCODE,                  Return code                   +
      RSNCODE),                 Reason code                   +
      MF=(E,PLIST)              -----
```

## BPX4TSP (tcsetpgrp) example

The following code sets the controlling terminal's foreground process group to a new value. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see [“tcsetpgrp \(BPX1TSP, BPX4TSP\) — Set the foreground process group ID ”](#) on page 944. AMODE 31 callers use [“BPX1TSP \(tcsetpgrp\) example”](#) on page 1305.

```
MVC     PROCID,..               Process group ID set by setpgrp
SPACE   ,
CALL    BPX4TSP,                Set foreground process group ID  +
      (=A(STDIN_FILENO),        Input: File descriptor          +
      PROCID,                   Input: Foreground process group ID+
      RETVAL,                   Return value: 0 or -1            +
      RETCODE,                  Return code                      +
      RSNCODE),                 Reason code                      +
      MF=(E,PLIST)              -----
```

## BPX4TST (tcsettables) example

The following code sets code page names, conversion tables and Code Page Change Notification (CPCN) capability. For the callable service, see [“tcsettables \(BPX1TST, BPX4TST\) — Set terminal code page names and conversion tables”](#) on page 946. For the data structure, see [“BPXYTCCP — Map the terminal control code page structure”](#) on page 1163. AMODE 31 callers use [“BPX1TST \(tcsettables\) example”](#) on page 1305.

```
XC      TCCP(TCCP#LENGTH),TCCP  Clear area
OI      TCCPFLAGB4,TCCPFASTP  Set local translation
MVC     TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
MVC     TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
MVC     TBLSOURCE,..           Initialize source conversion table
MVC     TBLTARGET,..          Initialize target conversion table
SPACE   ,
CALL    BPX4TST,                Set code page names and tables  +
      (=A(STDIN_FILENO),        Input: File descriptor          +
      =A(TCCP#LENGTH),          Input: Length of BPXYTCCP      +
      TCCP,                     Termcp structure, BPXYTCCP      +
      TBLSOURCE,                Source conversion table          +
      TBLTARGET,                Target conversion table          +
      RETVAL,                   Return value: 0 or -1            +
      RETCODE,                  Return code                      +
      RSNCODE),                 Reason code                      +
      MF=(E,PLIST)              -----
```

## BPX4TYN (ttyname) example

The following code retrieves the pathname for the standard error output file. For the callable service, see “[ttyname \(BPX1TYN, BPX4TYN\) \(POSIX version\) — Get the name of a terminal](#)” on page 956. AMODE 31 callers use “[BPX2TYN \(ttyname\) example](#)” on page 1306.

```

MVC  BUFLINA,=A(1023)      Maximum pathname
CALL BPX4TYN,              Determine terminal name      +
    (=A(STDERR_FILENO),    Input: File descriptor      +
    BUFLINA,               Length of buffer for pathname +
    BUFFERA,               Buffer for pathname of terminal +
    RETVAL,                Return value: 0, -1          +
    RETCODE,               Return code: describes why VAL=-1 +
    RSNCODE),              Reason code: qualifier on RETCODE +
    MF=(E,PLIST)           -----

```

## BPX4UMK (umask) example

The following code changes the process's file mode creation mask (to user read, group execute, other execute). For the callable service, see “[umask \(BPX1UMK, BPX4UMK\) — Set the file mode creation mask](#)” on page 960. For the data structure, see “[BPXYMODE — Map the mode constants](#)” on page 1106. AMODE 31 callers use “[BPX1UMK \(umask\) example](#)” on page 1307.

```

XC    S_MODE,S_MODE
MVI   S_MODE3,S_IXUSR+S_IXGRP+S_IXOTH  Search permission
SPACE
CALL  BPX4UMK,              Set file creation mask      +
    (S_MODE,               Input: Mode                BPXYMODE +
    RETVAL),               Return value: previous mode mask +
    MF=(E,PLIST)           -----

```

## BPX4ULA (unlinkat) example

The following code removes path name `usr/dataproc/next.t` from the directory that was specified by `DIRFD`. For the callable service, see “[unlinkat \(BPX1ULA, BPX4ULA\) — Remove a directory or directory entry relative to a directory file descriptor](#)” on page 969. AMODE 31 callers use “[BPX1ULA \(unlinkat\) example](#)” on page 1306.

```

MVC  BUFFERA(19),=CL19'usr/dataproc/next.t'
MVC  BUFLINA,=F'19'
MVC  DIRFD,..            Directory descriptor from opendir
MVC  FLAGS,=F'0'         Flags defined in BPXYFCTL
SPACE
CALL  BPX4ULA,            Remove a directory entry      +
    (BUFLINA,             Input: Pathname length      +
    BUFFERA,              Input: Pathname             +
    DIRFD,                Input: Directory descriptor  +
    FLAGS,                Input: Flags                +
    RETVAL,               Return value: 0 or -1        +
    RETCODE,              Return code                 +
    RSNCODE),             Reason code                 +
    VL,MF=(E,PLIST)       -----

```

## BPX4UMT (umount) example

The following code removes virtual file system `TESTLIB.FILESYS1` from the file tree. For the callable service, see “[umount \(BPX1UMT, BPX4UMT\) — Remove a virtual file system](#)” on page 961. For the data structure, see “[BPXYMTM — Map the modes for mount and unmount](#)” on page 1110. AMODE 31 callers use “[BPX1UMT \(umount\) example](#)” on page 1307.

```

MVC  FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC    MTM(MTM#LENGTH),MTM

```

## BPX4UNA (uname) example

```

MVI    MTM1,MTMUMOUNT      Unmount request
SPACE ,
CALL   BPX4UMT,             Remove a virtual file system +
      (FSNAME,             Input: File system name (44 char) +
      MTM,                 Input: Flags, BPXYMTM +
      RETVAL,              Return value: 0 or -1 +
      RETCODE,             Return code +
      RSNCODE),            Reason code +
      MF=(E,PLIST)         -----
```

## BPX4UNA (uname) example

The following code obtains information about the system on which the invoker is running. For the callable service, see “[uname \(BPX1UNA, BPX4UNA\) — Obtain the name of the current operating system](#)” on page 964. For the data structure, see “[BPXYUTSN — Map the response structure for uname](#)” on page 1172. AMODE 31 callers use “[BPX1UNA \(uname\) example](#)” on page 1307.

```

LA     R15,UTSN
STG    R15,UTSNA
SPACE ,
CALL   BPX4UNA,             Identify system +
      (UTSNL,              Input: Length of required buffer +
      UTSNA,              Output: ->UTSN BPXYUTSN +
      RETVAL,             Return value: -1 or >-1 +
      RETCODE,            Return code +
      RSNCODE),           Reason code +
      MF=(E,PLIST)         -----
```

## BPX4UNL (unlink) example

The following code removes path name `usr/dataproc/next.t` from the system. For the callable service, see “[unlink \(BPX1UNL, BPX4UNL\) — Remove a directory entry](#)” on page 966. AMODE 31 callers use “[BPX1UNL \(unlink\) example](#)” on page 1308.

```

MVC    BUFFERA(19),=CL19'usr/dataproc/next.t'
MVC    BUFLINA,=F'19'
SPACE ,
CALL   BPX4UNL,             Remove a directory entry +
      (BUFLINA,           Input: Pathname length +
      BUFFERA,            Input: Pathname +
      RETVAL,             Return value: 0 or -1 +
      RETCODE,            Return code +
      RSNCODE),           Reason code +
      MF=(E,PLIST)         -----
```

## BPX4UPT (unlockpt) example

The following code unlocks the subsidiary pseudoterminal device associated with the manager pseudoterminal to which the file descriptor refers. For the callable service, see “[unlockpt \(BPX1UPT, BPX4UPT\) — Unlock a pseudoterminal manager and subsidiary pair](#)” on page 973. AMODE 31 callers use “[BPX1UPT \(unlockpt\) example](#)” on page 1308.

```

CALL   BPX4UPT,             Unlocks subsidiary pty from manager +
      (MASTER_FD,         Input: File descriptor +
      RETVAL,             Return value: 0 or -1 +
      RETCODE,            Return code +
      RSNCODE),           Reason code +
      MF=(E,PLIST)         -----
```

## BPX4UQS (unquiesce) example

The following code unquiesces `TESTLIB.FILESYS1`, making its files available for use again. For the callable service, see “[unquiesce \(BPX1UQS, BPX4UQS\) — Unquiesce a file system](#)” on page 975. For the data

structure, see [“BPXYMTM — Map the modes for mount and unmount”](#) on page 1110. AMODE 31 callers use [“BPX1UQS \(unquiesce\) example”](#) on page 1308.

```

MVC  FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC   MTM(MTM#LENGTH),MTM    Zero MTM = don't force unquiesce
SPACE ,
CALL  BPX4UQS,                Unquiesce a file system      +
      (FSNAME,                Input: File system name (44 char) +
      MTM,                    Input: Flags, BPXYMTM          +
      RETVAL,                 Return value: 0 or -1           +
      RETCODE,                Return code                     +
      RSNCODE),               Reason code                     +
      MF=(E,PLIST)            -----

```

## BPX4UNS (unshare) example

The following code disassociates the current process from its IPC namespace and associates it with a newly created IPC namespace. For the callable service, see [“unshare \(BPX1UNS, BPX4UNS\) – Disassociate from shared execution resources](#). For the flag definitions, see [“BPXYCLNP— Map clone syscall parameters”](#) on page 1062. For the AMODE 31 example, see [“BPX1UNS \(unshare\) example”](#) on page 1308.

```

      CALL  BPX4UNS,                +
      (=A(CLONE_NEWIPC),           +
      RETVAL,                       +
      RETCODE,                       +
      RSNCODE),                     +
      MF=(E,PLIST)                  +
L     R15,RETVAL                    Load return value
C     R15,=F'-1'                    Test for -1 return
BE    PSEUDO                        Branch on error
...

```

## BPX4UTI (utime) example

The following code changes the access and modification times of /usr/private/workfile.t to the current time. For the callable service, see [“utime \(BPX1UTI, BPX4UTI\) — Set file access and modification times”](#) on page 981. AMODE 31 callers use [“BPX1UTI \(utime\) example”](#) on page 1309.

```

MVC  BUFFERA(23),=CL23'/usr/private/workfile.t'
MVC  BUFLINA,=F'23'
MVC  NEWTIMES,=X'FFFFFFFFFFFFFFFF'    Current time
SPACE ,
CALL  BPX4UTI,                Set file access and modify times +
      (BUFLINA,                Input: Pathname length          +
      BUFFERA,                 Input: Pathname                 +
      NEWTIMES,                Input: Access/Modification time  +
      RETVAL,                 Return value: 0 or -1             +
      RETCODE,                Return code                       +
      RSNCODE),               Reason code                       +
      MF=(E,PLIST)            -----

```

## BPX4WAT (wait) example

The following code waits for any of its children to end or stop. For the callable service, see [“wait \(BPX1WAT, BPX4WAT\) — Wait for a child process to end”](#) on page 984. For the data structure, see [“BPXYWAST — Map the wait status word”](#) on page 1172 and [“BPXYCONS — Constants used by services”](#) on page 1062. AMODE 31 callers use [“BPX1WAT \(wait\) example”](#) on page 1309.

```

LA     R15,WAST                Resolve address of STATUS
STG    R15,WASTA               Save address of STATUS
MVC    PROCID,=F'-1'           Wait for any child
SPACE ,
CALL   BPX4WAT,                Wait for a child process to end  +

```

## BPX4WLM ( \_\_WLM) example

(PROCID,	Input: PID being waited on	+	
=A(WNOHANG),	Input: options	BPXYCONS	+
WASTA,	->Exit status field,	BPXTWAST	+
RETVAL,	Return value: -1, 0, child PID		+
RETCODE,	Return code		+
RSNCODE),	Reason code		+
MF=(E,PLIST)	-----		

## BPX4WLM ( \_\_WLM) example

The following code connects to WLM as a work manager for the WEB subsystem type and WEB1 subsystem name. For the callable service, see “[\\_\\_wlm \(BPX1WLM, BPX4WLM\) — WLM interface service](#)” on page 1018. AMODE 31 callers use “[BPX1WLM \( \\_\\_WLM\) example](#)” on page 1309.

LA	R8,BUFFERA	Storage for _WWC	
USING	_WWC,R8	WLM_CONNECT_WORKMGR DSECT	
STG	R8,INARGLISTPTR	->_WWC list of parameters	
MVC	SUBSYSTYPE,=CL4'WEB '	WEB Subsystem Type	
MVC	SUBSYSNAME,=CL8'WEB1 '	WEB1 Subsystem Name	
LA	R15,SUBSYSTYPE		
STG	R15,_WWC_SUB_SYS	Pointer to Subsystem Type	
LA	R15,SUBSYSNAME		
STG	R15,_WWC_SUB_SYS_NM	Pointer to Subsystem Name	
SPACE	,		
CALL	BPX4WLM,	work_load_manager system call	+
	(=A(WLM_CONNECT_WORKMGR),	Input: Fcn Codes in BPXYWLM	+
	INARGLISTPTR,	Input: ->list of parameters	+
	RETVAL,	Return value: Varies with fcn code	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	MF=(E,PLIST)	-----	
DROP	R8		

## BPX4WRT (write) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). For the callable service, see “[write \(BPX1WRT, BPX4WRT\) — Write to a file or a socket](#)” on page 1032. AMODE 31 callers use “[BPX1WRT \(write\) example](#)” on page 1310.

*	MVC	FILEDESC,	File descriptor from open	
	MVC	BUFLINA,=F'80'		
	LA	R15,BUFFERA		
	STG	R15,BUFA		
	SPACE	,		
	CALL	BPX4WRT,	Write to a file	+
		(FILEDESC,	Input: File descriptor	+
		BUFA,	Input: ->Buffer	+
		PRIMARYALET,	Input: Buffer ALET	+
		BUFLINA,	Input: Number of bytes to write	+
		RETVAL,	Return value: -1 or bytes written	+
		RETCODE,	Return code	+
		RSNCODE),	Reason code	+
		MF=(E,PLIST)	-----	

## BPX4WRV (writev) example

The following code issues a writev for a socket. SOCKDESC was returned from a previous call to either BPX4SOC or BPX4ACP. For the callable service, see “[writev \(BPX1WRV, BPX4WRV\) — Write data from a set of buffers](#)” on page 1036. For the data structures, see “[BPXYSOCK — Map SOCKADDR structure and constants](#)” on page 1151 and “[BPXYIOV — Map the I/O vector structure](#)” on page 1098. AMODE 31 callers use “[BPX1WRV \(writev\) example](#)” on page 1310.

MVC	BUFFERA(16),=CL16'Here is the data'
LA	R2,BUFFERA
STG	R2,IOV_BASE
MVI	IOV_LEN,16



```

*      CALL BPX4WRV,          Write from a vector of buffers  +
      (SOCKDESC,             Input: Socket Descriptor        +
      =A(1),                 Input: Single element in iov    +
      IOV,                   Input: Iov containing info      +
      PRIMARYALET,           Input: Alet where iov resides    +
      PRIMARYALET,           Input: Alet of buffers for data  +
      RETVAL,                Return value: 0 or -1            +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                     +
      MF=(E,PLIST)           -----

```

## BPX4WTE (wait extension) example

The following code uses the #WAIT3 function to wait for any of its children to end or stop. For the callable service, see [“wait-extension \(BPX1WTE, BPX4WTE\) — Obtain status information for children”](#) on page 987. For the data structures, see [“BPXYWAST — Map the wait status word”](#) on page 1172 and [“BPXYRLIM — Map the rlimit, rusage, and timeval structures”](#) on page 1142 and [“BPXYCONS — Constants used by services”](#) on page 1062. AMODE 31 callers use [“BPX1WTE \(wait extension\) example”](#) on page 1310.

```

LA      R15,WAST              Resolve address of WAST
STG     R15,WASTA             Save address of WAST
LA      R15,RUSAGE           Resolve address of RUSAGE
STG     R15,RUSAGEA          Save address of RUSAGE
SPACE ,
CALL    BPX4WTE,              Wait for a child process to end  +
      (=A(#WAIT3),           Input: function      BPXYCONS  +
      0,                     Input: id type        +
      0,                     Input: id             +
      WASTA,                 ->Exit status field, BPXTWAST +
      =A(WNOHANG),           Input: options      BPXYCONS  +
      RUSAGEA,               ->Rusage structure, BPXYRLIM  +
      RETVAL,                Return value: -1, 0, child PID +
      RETCODE,               Return code            +
      RSNCODE),              Reason code            +
      MF=(E,PLIST)           -----

```

## Reentrant return linkage

```

XGR     R15,R15              Zero return code
L        R0,@SIZEDAT          Size this program's getmain area
LGR     R1,R13               R1 -> this program's getmain area
LG      R13,@BACK            R2 -> caller's save area
DROP    R13
FREEMAIN RU,LV=(0),A=(1)
LG      R14,8(R13)           Restore caller's R14
LMG     R0,R12,16(R13)       Restore caller's R0-R12
BR      R14                  Branch back to caller

```

```

SPACE , * * * * * * * * * * Program constants * * * * *
@SIZEDAT DC A(@ENDSTOR-@STORE) Size of this getmain storage
MNTEL    DC A(MNTEH#LENGTH+MNTEH#LENGTH)
*
PGPSL    DC A(PGPS#LENGTH)    Length of PGPS structure
RMONL    DC A(RMON#LENGTH)    Length of RMON structure
SSTFL    DC A(SSTF#LENGTH)    Length of SSTF structure
STATL    DC A(STAT#LENGTH)    Length of STAT structure
UTSNL    DC A(UTSN#LENGTH)    Length of UTSN structure
SPACE ,
PRIMARYALET DC A(0)           Primary ALET

```

```

* * * * * * * * * * * * * * * * Program getmain variables * * *
DS       0D
ACPSOCK  DS F                 Accepted socket descriptor
ADDR_INFO_PTR DS F           ->Addr_Info Structure
APPLENV  DS CL8               Application Environment
APPLDATALEN DS F             Application Data Length
APPLDATAPTR DS A              -> Application Data

```

## Reentrant return linkage

ARGCNT	DS	F	Argument count
ARGLLST	DS	3AD	Argument lengths list
ARGSLST	DS	3AD	Arguments list
ATTRIBUTES	DS	F	Attributes
ATTADDR	DS	AD	Shared memory attached address
BUFA	DS	FD	->buffer
BUFB	DS	FD	->buffer
BUFFR	DS	F	
BUFCNTB	DS	F	Count associated with BUFFERB
BUFFERA	DS	CL1024	Utility buffer A, length 1024
BUFFERB	DS	CL1024	Utility buffer B, length 1024
BUFLENA	DS	F	Number of bytes used in buffer A
BUFLNB	DS	F	Number of bytes used in buffer B
BUFW	DS	F	Number of words used in BUF
BYTERECD	DS	F	Bytes Received
CANONICAL_LENGTH	DS	F	Canonical name length
CELLUUID	DS	CL36	Cell UUID (string form)
CLASSIFYLEN	DS	F	Classify Area Length
CLASSIFYAREAPTR	DS	A	-> Classify Area
CLSLEN	DS	F	Class name length
CLS	DS	CL8	Class name
COMMAND	DS	F	User defined command
CONMSGTYPE	DS	F	Console msg type (modify or stop)
DIRECTDES	DS	F	Directory descriptor
ECB01	DS	F	Event Control Block # 1
ECB02	DS	F	Event Control Block # 2
EGID	DS	F	User ID
ENT	DS	CL40	Entity name
ENTLEN	DS	F	Entity name length
ENTRYPT	DS	FD	Entry Point
ENVCNT	DS	F	Number of environment variables
ENVLENS	DS	FD	Length of environment variables
ENVPARMS	DS	FD	Environment variables
EPADDR	DS	AD	Entry point address
EUID	DS	F	User ID
EVENTLIST	DS	A	Event list for thread posting
EXITRTNA	DS	AD	Exit routine address
EXITPLA	DS	AD	Exit Parm list address
FDCNT	DS	F	File descriptor count
FDLST	DS	F	File descriptor list
FDLISTPTR	DS	A	-> File Descriptor List
FILEDESC	DS	F	File descriptor
FILEDES2	DS	F	File descriptor
FLAGS	DS	F	Utility Flags word
FSNAME	DS	CL44	File system name
FSTYPE	DS	CL8	File system type
GRNAMELN	DS	F	Group name length
GROUP	DS	F	Group
GROUPCNT	DS	F	Group count
GROUPID	DS	F	Group ID (PID of group leader)
GRPGMNAME	DS	CL8	Group program name
HINTS_PTR	DS	FD	->Addr_Info Structure
HOST_BUFFER	DS	CL255	Host buffer (up to 255 Characters)
HOST_BUFFER_LENGTH	DS	F	Host buffer length
IDENT	DS	CL8	Identity of new thread security
INARG	DS	F	Input argument
INARGLIST	DS	A	Input arglist
INARGLISTPTR	DS	AD	Input arglist ptr
INCR	DS	F	Increment value for nice
INTMASK	DS	XL8	Signal mask
INITADDR	DS	FD	Address __map_init parm list
INITPARM	DS	OC	__map_init parm list
	ORG	**_MMG_INIT_PARM_LEN	
INITRTNA	DS	AD	->Initialization routine
INTRSTATE	DS	A	Interrupt state
INTRTYPE	DS	A	Interrupt type
ITIMA	DS	AD	->BPXYITIM structure
KEY	DS	F	Interprocess Communication KEY
LIBPTHLN	DS	F	Library Path Length (BPX4LOD)
LIBPATH	DS	CL100	Library Path (BPX4LOD)
LOCKADDR	DS	AD	->Lockword
LOCKTOKENADDR	DS	AD	->LockToken
LOCKATTRADDR	DS	AD	->LockAttr
LOCKWORD	DS	F	Lockword (BPX4SLK)
LSOCKADR	DS	F	Local socket structure
LTOKEN	DS	CL8	Local token
MAP_ADDRESS	DS	AD	->mapped area
MAP_LENGTH	DS	FD	length of mapped area
MASTER_FD	DS	F	Master file descriptor
MSG_ID	DS	F	IPC Message Queue ID
MSGATTRLEN	DS	F	Length of BPX4CCA
MSGATTR	DS	CL100	Storage for BPX4CCA

MODSTRINGPTR	DS	F	Address of user msg buffer
MODIFYSTGLEN	DS	F	Length of user msg buffer
NANOSECONDS	DS	F	Count of nanoseconds
NCATCHER	DS	A	New catcher
NEWFLAGS	DS	F	New flags
NEWHANDL	DS	FD	New Handler
NEWLEN	DS	XL8	Length file
NEWMASK	DS	XL8	New mask for signals
NEWMASKA	DS	A	->New mask
NEWPASS	DS	CL8	Password
NEWPASSLEN	DS	F	Password length
NEWTIMES	DS	DL2	New access/modification time
NODE_NAME	DS	CL255	Node Name (up to 255 Characters)
NODE_NAME_LENGTH	DS	F	Node Name Length
NUMB_SEMS	DS	F	IPC Number of semaphores in set
NUMB_SEM_OPS	DS	F	IPC Number of semaphore ops
OCATCHER	DS	A	Old catcher
OFFSET	DS	CL8	File offset
OLDHANDL	DS	FD	Old handler
OLDFLAGS	DS	F	Old flags
OLDMASK	DS	CL8	Old signal mask
OLDMASKA	DS	A	->Old mask
OLDPASS	DS	CL8	Password
OLDPASSLEN	DS	F	Password length
OPTIONS	DS	F	Options
PARALLELEU	DS	F	Parallel Eu
PASSWORD	DS	CL8	Password
PGMNAME	DS	CL8	Program name
PGMNAMEL	DS	F	Length PGMNAME
PLIST	DS	13A	Max number of parms
PRINUID	DS	CL36	Principal UID (string form)
PRIORITY	DS	F	Priority value
PROCID	DS	F	Process ID
PROCTOK	DS	F	Relative process number
PT_NEWA	DS	AD	Address of PT_NEW
PT_OLD	DS	CL66	Pthread tag - old
PT_OLDA	DS	AD	Address of PT_OLD
PT_OLDL	DS	F	Length of tag in PT_NEW
READFD	DS	F	File descriptor - input file
REFPT	DS	F	File reference point
RESOURCE	DS	F	Resource
RESULTS_PTR	DS	FD	->Addr_Info Structure
RETCODE	DS	F	Return code (ERRNO)
RETURNEDADDRESS	DS	AD	Returned address in doubleword
RETVAL	DS	F	Return value (0, -1 or other)
RETVAL64	DS	FD	64-bit return value
RGID	DS	F	User ID
RSOCKADR	DS	F	Remote socket structure
RUID	DS	F	User ID
RUSAGEA	DS	AD	->Rusage
RSNCODE	DS	F	Reason code (ERRNOJR)
SECONDS	DS	F	Time in seconds
SEGADDR	DS	AD	IPC Shared Memory segment Addr
SELLIST	DS	F	List to use for select calls
SEM_ID	DS	F	IPC Semaphore set ID
SEM_NUMBER	DS	F	IPC Semaphore number
SERVICE_BUFFER	DS	CL32	Service Buffer ( up to 32 Characters)
SERVICE_BUFFER_LENGTH	DS	F	Service buffer length
SERVICE_NAME	DS	CL32	Service Name (up to 32 Characters)
SERVICE_NAME_LENGTH	DS	F	Service Name Length
SHM_ID	DS	F	IPC Shared Memory segment ID
SIGNAL	DS	A	Signal
SIGNALREG	DS	AD	Signal registration, user data
SIGNALOPTIONS	DS	A	Signal options
SIGPID	DS	F	Signal processs id for BPX4PAF
SIGRET	DS	CL8	Signal return mask
SIRTN	DS	AD	Signal interrupt routine
SMF_TYPE	DS	F	SMF record type
SMF_SUBTYPE	DS	F	SMF record subtype
SOCKADDR_LENGTH	DS	F	Length of SockAddr
SOCKETS	DS	0XL8	Socket vector for socket call
SOCKDESC	DS	F	Socket descriptor
SOCKDESC2	DS	F	Second Socket descriptor
SRVCADDR	DS	FD	Address __map_service parm list
SRVCPARM	DS	0C	__map_service parm list
	ORG	**+3*_MMG_SERVICE_PARM_LEN	Room for three entries
STATFLD	DS	AD	Status field
STATUS	DS	F	Status
STATUSA	DS	A	->STATUS
SUBSYSTYPE	DS	CL4	Subsystem Type
SUBSYSNAME	DS	CL8	Subsystem Name
TARPID	DS	F	Target processs id for BPX4PAF

## Reentrant return linkage

```

      ORG    BUFFERB      remap utility buffer B
TBLSOURCE DS    XL256      Source conversion table
TBLTARGET DS    XL256      Target conversion table

      ORG
TERMMASK  DS    XL8        Signal termination mask
THID      DS    XL8        Thread ID
TOKEN     DS    F          Relative IPC member or Misc Token
TRXCLASS  DS    CL8        Transaction Class
USERID    DS    F          User ID
USERDATA  DS    FD         User Data
USERNAME  DS    CL8        User name
USERNLEN  DS    F          Length USERNAME
HOST_NAME DS    CL8        HOST name
HOST_NAMELEN DS    F        Length HOST_NAME
HOST_ADDR DS    CL8        HOST IP address
HOST_ADDRLEN DS    F        Length HOST_ADDR
HOSTENT_PTR DS    FD        Length HOST_ADDR
USERWORD  DS    FD         User data
WAITMASK  DS    F          Mast for signal waits
WHO        DS    F          Who for rusage
WRITEFD   DS    F          File descriptor - output file
LFUIOPTR  DS    FD         Pointer to FUIO structure
@ENDSTOR  EQU    *          End of getmain storage
      SPACE 3 ,
      IVTBUFL
      SPACE 3 * * * * * Register equates * * * * *
R0        EQU    0
R1        EQU    1          Parameter list pointer
R2        EQU    2
R3        EQU    3
R4        EQU    4
R5        EQU    5
R6        EQU    6
R7        EQU    7
R8        EQU    8
R9        EQU    9
R10       EQU    10         Second getmain storage register
R11       EQU    11         Second program base register
R12       EQU    12         Program base register
R13       EQU    13         Savearea and getmain storage base
R14       EQU    14         Return address
R15       EQU    15         Branch location
      SPACE 3 * * * * * External * * * * *
      SPACE ,
      EXTRN EXITRTN
      EXTRN SIRTN
      END

```

## Appendix F. Examples of nonreentrant entry linkage

### Example of nonreentrant entry linkage—AMODE 31

This example shows the function for the `__getthent` service in a nonreentrant program. For a reentrant example of `__getthent`, see “BPX1GTH (`__getthent`) example” on page 1251. For an example of reentrant entry and return linkage, see Appendix D, “Callable services examples—AMODE 31,” on page 1219 and “Reentrant return linkage” on page 1311.

```

BPXB1SM5 CSECT , Nonreentrant linkage
BPXB1SM5 AMODE 31
BPXB1SM5 RMODE ANY
        USING *,R15 Program addressability
@BEGIN0 B @BEGIN1 Branch around program header
        DC C'BPXB1SM5 - nonreentrant __getthent invoker'
        DS 0H
@BEGIN1 STM R14,12,12(R13) Save callers registers
        ST R13,@BACK Save ->Callers save area
        LA R13,@SAVE00 Program addressability
        DROP R15
        USING @SAVE00,R13 Program addressability
        B @BEGIN2
@SAVE00 DS 0D Standard save area - 72 Bytes
        DS A
@BACK DS A Backwards save area pointer
@FORWARD DS A Forwards save area pointer
        DS 15A Regs 14,15,0-12
RETURN XR R15,R15 Zero return code
RETURNRC L R13,@BACK Restore callers r13
        L R14,12(,R13) Restore callers r14
        LM R0,R12,20(R13) Restore callers r0-r12
        BSM 0,R14 Branch back to caller
R0 EQU 0
R1 EQU 1 Parameter list pointer
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13 Program and save area base
R14 EQU 14 Return address
R15 EQU 15 Branch location
@BEGIN2 EQU * * * * * End of the entry linkage code
        EJECT
        LA R5,BUFFERA R5-> Input buffer
        ST R5,PGTHAB -> input buffer
        USING PGTHA,R5 R5 base for PGTHA
        XC PGTHA,PGTHA Null input area
        MVI PGTHAFLAG1,PGTHAPROCESS+PGTHATHREAD
        MVI PGTHAID,PGTH#FIRST First thread
        LA R15,BUFFERB Pgthb, Output buffer
        ST R15,PGTHBB Output Buffer
        SPACE , * * * * *
        LA R0,=CL8'BPX1GTH ' LOAD -> entry point name
        XR R1,R1 No JOBLIB or LINKLIB DCB
        SVC 8 Issue LOAD SVC
        ST R0,GETENTRY Store BPX1GTH entry point
        L R15,GETENTRY Address of BPX1GPS load module
        CALL (15), Get process data +
                (PGTHAL, Length of buffer +
                PGTHAB, Buffer, mapped by BPXPGPHA +
                PGTHBL, Length of output buffer +
                PGTHBB, Buffer, mapped by BPXPGTHC +
                RETVAL, Return value (next, eof or error) +
                RETCODE, Return code +
                RSNCODE), Reason code +
        VL -----

```

## Example of nonreentrant entry linkage—AMODE 31

	SPACE	, * * * * *	
	L	R15,RETVAL	Load return value
	C	R15,=F'-1'	Test for -1 return
	BE	RETURNRC	-1 is error
	SPACE	, * * * * *	Initialize WTO area & message
	MVI	XPID,C' '	Blank out variable portion of msg ge
	MVC	XPID+1(WTO#BLANK-1),XPID	
	SPACE	, * * * * *	Process ID to printable hex
	LA	R6,BUFFERB	R6-> Output buffer
	ST	R6,PGTHBB	-> output buffer
	USING	PGTHB,R6	R6 base for PGTHB
	L	R8,PGTHBPID	R8 = process ID
	LA	R9,XPID	To be placed at message start
	LA	R15,8	8 nibbles to convert (4 bytes)
NIBBLE	LA	R10,9	For 0-9 / A-F compare
	LR	R11,R8	Target bits in 0-3      XXXXXXXZ
	SRL	R11,28	Bits 0-3 to 28-31      0000000X
	SLL	R8,4	Drop bits 0-3 off end      YYYYYYZ0
	CLR	R11,R10	Are 4 bits 0-9 or A-F
	BC	B'0010',AF	Branch if A-F
AF	LA	R11,57(,R11)	Add for 0-9 (57+183=240 or F0)
	LA	R11,183(,R11)	Add for 0-F (183+10=193 or C1)
	STC	R11,0(,R9)	Store to results location
	LA	R9,1(,R9)	Increment R9 to next location
	BCT	R15,NIBBLE	Decrement half byte counter, loop
	SPACE	, * * * * *	Test status bits
* Go after the		state of the process	
	LA	R7,PGTHB	
	SLR	R9,R9	
	ICM	R9,7,PGTHBOFFC	
	AR	R7,R9	
	USING	PGTHC,R7	
	LA	R8,PGTHB	
	SLR	R9,R9	
	ICM	R9,7,PGTHBOFFJ	
	AR	R8,R9	
	USING	PGTHJ,R8	
	MVI	THREAD,C'1'	Assume single
	TM	PGTHCFLAG1,PGTHCMULPROCESS	if multiprocess
	BZ	NOTMULT	
NOTMULT	MVI	THREAD,C'M'	
	MVC	STATE,PGTHJSTATUS2	Z, W, X, S, C, F, K, R ...
	TM	PGTHCFLAG1,PGTHCSWAP	if swapped out
	BZ	NOTSWAP	
	MVC	SWAPA,=CL4'SWAP'	
NOTSWAP	TM	PGTHCFLAG1,PGTHCSTOPPED	if stopped
	BZ	NOTSTOP	
	MVC	STOPA,=CL4'STOP'	
NOTSTOP	TM	PGTHCFLAG1,PGTHCTRACE	if ptrace
	BZ	NOTTRAC	
	MVC	TRACA,=CL4'TRAC'	
NOTTRAC	EQU	*	
	SPACE	, * * * * *	Display message to operator
	LA	R2,WTOAREA	R2->WTO message area
	WTO	TEXT=(R2)	Write to Operator
	SPACE	, * * * * *	Loop back
	MVC	PGTHACONTINUE,PGTHBCONTINUE	get next thread
	B	GETTH	
WTOAREA	DS	0F	WTO message
	DC	AL2(WTO#LENGTH)	Length of area
	DC	CL4'PID='	Process ID =
XPID	DS	CL8	Hex of process ID
	DS	CL1	
THREAD	DS	CL1	1, M or H
	DS	CL1	
STATE	DS	CL1	Z, W, X, C, F, K, R ...
	DS	CL1	
SWAPA	DS	CL4	SWAP or blank
	DS	CL1	
STOPA	DS	CL4	STOP or blank
	DS	CL1	
TRACA	DS	CL4	TRAC or blank
WTO#BLANK	EQU	*-XPID	Length to blank
	DC	C'.'	
WTO#LENGTH	EQU	*-WTOAREA	Length of WTO area
	SPACE	,	
GETENTRY	DS	A	Address of BPX1GPS
RETVAL	DS	F	Return value - next
RETCODE	DS	F	Return code
RSNCODE	DS	F	Reason code
	SPACE	,	
BUFFERA	DS	CL50	Buffer for Process data

BUFFERB	DS	CL500	Buffer for Process data
PGTHAL	DC	A(PGTHA#LEN)	Length of PGTH buffer
PGTHAB	DS	A(PGTHA)	->Process data buffer
PGTHBL	DC	A(500)	Length of PGTH buffer
PGTHBB	DS	A(PGTHB)	->Process data buffer
BPXYPGTH	DSECT=NO		Place in current CSECT / DSECT
END			

## Example of nonreentrant entry linkage—AMODE 64

This example shows the function for the `__getthent` service in a nonreentrant program. For a reentrant example of `__getthent`, see “BPX4GTH (`__getthent`) example” on page 1344. For an example of reentrant entry and return linkage, see Appendix E, “Callable services examples—AMODE 64,” on page 1313 and “Reentrant return linkage” on page 1403.

BPXB1SM6	CSECT	,	Nonreentrant linkage
BPXB1SM6	AMODE	64	
	SYSSTATE	AMODE64=YES	
@BEGIN0	J	@BEGIN1	Branch around program header
	DC	C'BPXB1SM6 - nonreentrant __getthent invoker'	
	DS	0H	
@BEGIN1	STMG	R14,12,12(R13)	Save callers registers
	BRAS	R12,PDATA1	Establish addressability save area ea
	DC	A(@SAVE00)	
PDATA1	L	R12,0(,R12)	
	USING	@SAVE00,R12	
	STG	R13,@BACK	Save ->Callers save area
	LA	R13,@SAVE00	Program addressability
	DROP	R12	
	USING	@SAVE00,R13	Program addressability
	J	@BEGIN2	
@SAVE00	DS	0D	Standard save area - 144 Bytes
	DS	A	Reserved
	DS	CL4'F4SA'	Linkage Type
	DS	15AD	Regs 14,15,0-12
@BACK	DS	AD	Backwards save area pointer
@FORWARD	DS	AD	Forwards save area pointer
RETURN	XR	R15,R15	Zero return code
RETURNRC	LG	R13,@BACK	Restore callers r13
	LG	R14,12(,R13)	Restore callers r14
	LMG	R0,R12,20(R13)	Restore callers r0-r12
	BR	R14	Branch back to caller
R0	EQU	0	
R1	EQU	1	Parameter list pointer
R2	EQU	2	
R3	EQU	3	
R4	EQU	4	
R5	EQU	5	
R6	EQU	6	
R7	EQU	7	
R8	EQU	8	
R9	EQU	9	
R10	EQU	10	
R11	EQU	11	
R12	EQU	12	
R13	EQU	13	Program and save area base
R14	EQU	14	Return address
R15	EQU	15	Branch location
@BEGIN2	EQU	* * * * *	End of the entry linkage code
	EJECT	,	
	LA	R5,BUFFERA	R5-> Input buffer
	STG	R5,PGTHAB	-> input buffer
	USING	PGTHA,R5	R5 base for PGTHA
	XC	PGTHA,PGTHA	Null input area
	MVI	PGTHAFLAG1,PGTHAPROCESS+PGTHATHREAD	
	MVI	PGTHAPID,PGTH#FIRST	First thread
	LA	R15,BUFFERB	Pgthb, Output buffer
	STG	R15,PGTHBB	Output Buffer
	SPACE	, * * * * *	
	LA	R0,=CL8'BPX4GTH '	LOAD -> entry point name
	XGR	R1,R1	No JOBLIB or LINKLIB DCB
	SVC	8	Issue LOAD SVC
	NILL	R0,X'FFFE'	Turn off low order bit
	STG	R0,GETENTRY	Store BPX4GTH entry point
GETTH	LG	R15,GETENTRY	Address of BPX4GTH load module
	CALL	(15),	Get process data
		(PGTHAL,	Length of buffer

## Example of nonreentrant entry linkage—AMODE 64

	PGTHAB,	Buffer, mapped by BPXYPGTH	+
	PGTHBL,	Length of output buffer	+
	PGTHBB,	Buffer, mapped by BPXYPGTH	+
	RETVAL,	Return value (next, eof or error)	+
	RETCODE,	Return code	+
	RSNCODE),	Reason code	+
	LINKINST=BALR		
	SPACE , * * * * *		
	L R15,RETVAL	Load return value	
	C R15,=F'-1'	Test for -1 return	
	BE RETURNRC	-1 is error	
	SPACE , * * * * *	Initialize WTO area & message	
	MVI XPID,C' '	Blank out variable portion of msg	ge
	MVC XPID+1(WTO#BLANK-1),XPID		
	SPACE , * * * * *	Process ID to printable hex	
	LA R6,BUFFERB	R6-> Output buffer	
	STG R6,PGTHBB	-> output buffer	
	USING PGTHB,R6	R6 base for PGTHB	
	L R8,PGTHBPID	R8 = process ID	
	LA R9,XPID	To be placed at message start	
	LA R15,8	8 nibbles to convert (4 bytes)	
	LA R10,9	For 0-9 / A-F compare	
NIBBLE	LR R11,R8	Target bits in 0-3	YYYYYYZ
	SRL R11,28	Bits 0-3 to 28-31	0000000X
	SLL R8,4	Drop bits 0-3 off end	YYYYYYZ0
	CLR R11,R10	Are 4 bits 0-9 or A-F	
	BC B'0010',AF	Branch if A-F	
	LA R11,57(,R11)	Add for 0-9 (57+183=240 or F0)	
AF	LA R11,183(,R11)	Add for 0-F (183+10=193 or C1)	
	STC R11,0(,R9)	Store to results location	
	LA R9,1(,R9)	Increment R9 to next location	
	BCT R15,NIBBLE	Decrement half byte counter, loop	
	SPACE , * * * * *	Test status bits	
* Go after the	state of the process		
	LA R7,PGTHB	Get the PGTHB address	
	SLR R9,R9	Clear r9	
	ICM R9,7,PGTHBOFFC	Get offset for PGTHC	
	AR R7,R9	Calculate address	
	USING PGTHC,R7	Addressability for PGTHC	
	LA R8,PGTHB	Get the PGTHB address	
	SLR R9,R9	Clear r9	
	ICM R9,7,PGTHBOFFJ	Get offset for PGTHJ	
	AR R8,R9	Calculate address	
	USING PGTHJ,R8	Addressability for PGTHJ	
	MVI THREAD,C'1'	Assume single	
	TM PGTHCFLAG1,PGTHCMULPROCESS	if multiprocess	
	BZ NOTMULT		
	MVI THREAD,C'M'		
NOTMULT	MVC STATE,PGTHJSTATUS2	Z, W, X, S, C, F, K, R ...	
	TM PGTHCFLAG1,PGTHCSWAP	if swapped out	
	BZ NOTSWAP		
	MVC SWAPA,=CL4'SWAP'		
NOTSWAP	TM PGTHCFLAG1,PGTHCSTOPPED	if stopped	
	BZ NOTSTOP		
	MVC STOPA,=CL4'STOP'		
NOTSTOP	TM PGTHCFLAG1,PGTHCTRACE	if ptrace	
	BZ NOTTRAC		
	MVC TRACA,=CL4'TRAC'		
NOTTRAC	EQU *		
	SPACE , * * * * *	Display message to operator	
	LA R2,WTOAREA	R2->WTO message area	
	WTO TEXT=(R2)	Write to Operator	
	SPACE , * * * * *	Loop back	
	MVC PGTHACONTINUE,PGTHBCONTINUE	get next thread, process	
	J GETTH		
WTOAREA	DS 0F	WTO message	
	DC AL2(WTO#LENGTH)	Length of area	
	DC CL4'PID= '	Process ID =	
XPID	DS CL8	Hex of process ID	
	DS CL1		
THREAD	DS CL1	1, M	
	DS CL1		
STATE	DS CL1	Z, W, X, C, F, K, R ...	
	DS CL1		
SWAPA	DS CL4	SWAP or blank	
	DS CL1		
STOPA	DS CL4	STOP or blank	
	DS CL1		
TRACA	DS CL4	TRAC or blank	
WTO#BLANK	EQU *-XPID	Length to blank	
	DC C'.'		
WTO#LENGTH	EQU *-WTOAREA	Length of WTO area	



	SPACE ,	
GETENTRY	DS AD	Address of BPX4GTH
RETVAL	DS F	Return value - next
RETCODE	DS F	Return code
RSNCODE	DS F	Reason code
	SPACE ,	
BUFFERA	DS CL50	Buffer for Process data
BUFFERB	DS CL500	Buffer for Process data
PGTHAL	DC A(PGTHA#LEN)	Length of PGTH buffer
PGTHAB	DS AD(PGTHA)	->Process data buffer
PGTHBL	DC A(500)	Length of PGTH buffer
PGTHBB	DS AD(PGTHB)	->Process data buffer
	BPXYPGTH DSECT=NO	Place in current CSECT / DSECT
	END	



## Appendix G. The relationship of z/OS UNIX signals to callable services

The signal information in this topic is needed by compiler writers who are implementing POSIX in a high-level language.

Signals support the following callable services:

- [“alarm \(BPX1ALR, BPX4ALR\) — Set an alarm ” on page 30](#)
- [“kill \(BPX1KIL, BPX4KIL\) — Send a signal to a process ” on page 360](#)
- [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#)
- [“mvunsigsetup \(BPX1MSD, BPX4MSD\) — Detach the signal setup ” on page 495](#)
- [“pause \(BPX1PAS, BPX4PAS\) — Suspend a process pending a signal” on page 539](#)
- [“ptrace \(BPX1PTR, BPX4PTR\) — Control another process for debugging” on page 620](#)
- [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action ” on page 834](#)
- [“sigpending \(BPX1SIP, BPX4SIP\) — Examine pending signals ” on page 843](#)
- [“sleep \(BPX1SLP, BPX4SLP\) — Suspend execution of a process for an interval of time ” on page 859](#)
- [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered” on page 648](#)
- [“sigprocmask \(BPX1SPM, BPX4SPM\) — Examine or change a process's signal mask ” on page 845](#)
- [“sigsuspend \(BPX1SSU, BPX4SSU\) — Change the signal mask and suspend the thread until a signal is delivered” on page 851](#)

### High-level-language signal interfaces

In addition to the signal interface callable services that are defined by POSIX, z/OS UNIX provides the following signal interface services:

#### **mvssigsetup service**

Sets up and defines the *signal interface routine (SIR)*. The SIR is a routine that is provided by the high-level language. For information about how to write the SIR and the interface to it, see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals” on page 487](#).

#### **mvunsigsetup service**

Detaches the interface to the SIR and returns the parameters set up in mvssigsetup. See [“mvunsigsetup \(BPX1MSD, BPX4MSD\) — Detach the signal setup ” on page 495](#).

#### **ptrace service**

Controls the running of another process for debugging programs. See [“ptrace \(BPX1PTR, BPX4PTR\) — Control another process for debugging” on page 620](#).

#### **queue\_interrupt service**

Returns the last signal delivered. See [“queue\\_interrupt \(BPX1SPB, BPX4SPB\) — Return the last interrupt delivered” on page 648](#).

These interfaces allow a runtime library (RTL) for a high-level language to control the flow of signals. Each high-level language defines its own linkage interface between callable procedures; for example, the C language has a linkage stack and register interface between function procedures, which are unique to C.

Delivery of signals involves:

- Interrupting a currently running procedure
- Saving the status of the code that was interrupted
- Invoking a callable procedure known as the *signal catcher*, or signal handler.

## How high-level languages use signals

Invoking a callable service involves setting up registers that are unique to the high-level language.

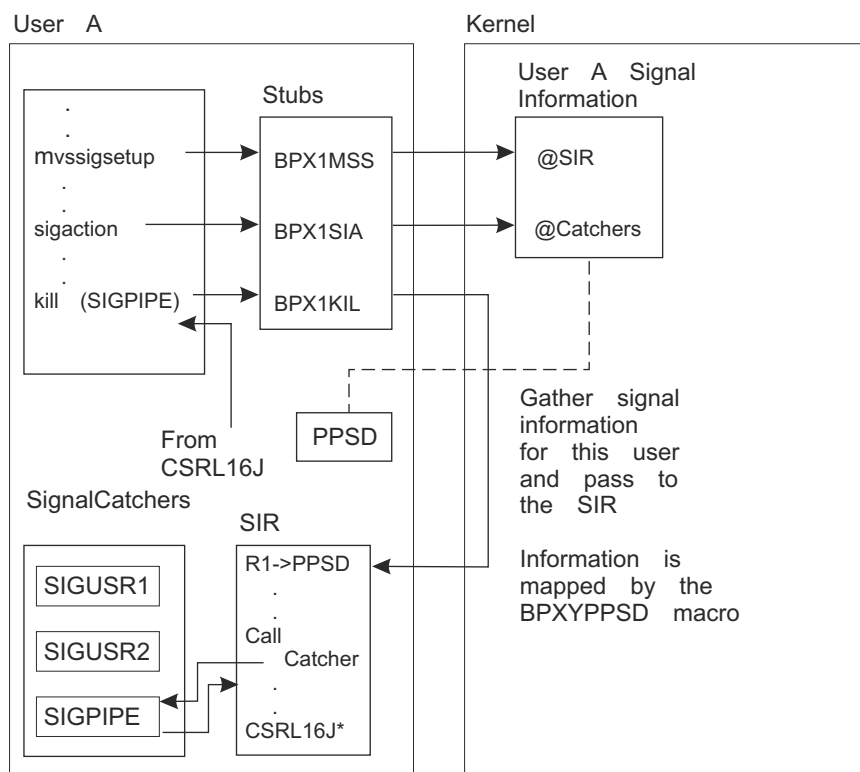
1. The RTL, using these callable services, sets up a SIR to receive control when a signal occurs.
2. The SIR procedure performs the necessary language linkages and POSIX functions to call the signal catcher procedure.
3. The signal catcher may return to the SIR.
4. The SIR performs the necessary language and POSIX functions to return to the interrupted procedure after the signal catcher returns.
5. The CSRL16J system service loads all registers and the PSW condition code and jumps to the instruction that was interrupted by the signal.

## Signal setup when linking to callable services

When a task invokes the first z/OS UNIX call, the address space (if needed) and task are set up for z/OS UNIX callable services. Setting up for z/OS UNIX callable services is known as dubbing the address space and dubbing the task. When an address space is dubbed, a new process is created and assigned a unique process ID.

A *dubbed task* is a thread that is assigned an 8-character thread ID. This thread ID is unique within the process. Threads in different processes could have the same thread ID. When the first z/OS UNIX call is made and the task is dubbed, the current program request block (PRB) that dubbed the task is also recorded. This not only dubs the task, but also sets it up for signals.

Figure 6 on page 1414 shows the flows for the various signal functions when a synchronous signal SIGPIPE is generated with the kill service.



\*CSRL16J is load 16 and Jump System Service.

Figure 6. Program flow of mvssigsetup and sigaction with signal interface routine (SIR)

For more information about the setup and use of SIRs, see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals”](#) on page 487. For more information about signal catchers, see [“sigaction \(BPX1SIA, BPX4SIA\) — Examine or change a signal action”](#) on page 834.

## ESPIE or ESTAE and the SIGILL, SIGFPE, and SIGSEGV signals

High-level languages generate the SIGILL, SIGFPE, and SIGSEGV signals. In z/OS UNIX, the kill service is invoked to generate these signals. The ESPIE or ESTAE must also use the kill service to generate SIGILL, SIGFPE, and SIGSEGV. High-level languages can define an ESPIE or ESTAE routine to receive control after an incorrect hardware instruction, arithmetic operation, or memory reference.

Since z/OS UNIX does not generate or process the SIGILL, SIGFPE, and SIGSEGV signals, it is the responsibility of the high-level language's RTL to define what happens when a signal catcher is defined for these signals and the signal catcher returns to the failing instruction. For information about how the compiler defines what happens in this case, see [Generating a signal in z/OS XL C/C++ Programming Guide](#).

ESPIE or ESTAE routines in high-level languages must also invoke the ptrace service. For more information about the ptrace service see [“ptrace \(BPX1PTR, BPX4PTR\) — Control another process for debugging”](#) on page 620.

### When signals are and are not supported

All signal functions are supported when the task is set up for signals, when it is running with the signal delivery key, and when its current program request block (PRB) is the same PRB as when the task was set up for signals. When this is not the case, some signal functions are not supported, or they function differently. [Table 37 on page 1415](#) defines these signal functions.

The mvssigsetup columns in [Table 37 on page 1415](#) describe a task that is set up with the mvssigsetup service. When a task invokes the mvssigsetup service, the current PRB is recorded for future signal delivery. When a task is set up for signals by mvssigsetup, signals are only delivered when the task's current PRB is the same PRB that called mvssigsetup.

<i>Table 37. Support of signal calls</i>					
Service	Task mvssigsetup		Not signal delivery key	Task not mvssigsetup	
	The current PRB called mvssigsetup	The current PRB did not call mvssigsetup		The current PRB dubbed the task	The current PRB did not dub the task
BPX1ALR	RV=Seconds	Abend	RV=Seconds	RV=Seconds	Abend
BPX1KIL	RV=0	RV=0	RV=0	RV=0	RV=0
BPX1MSD	RV=0	RV=0	RV=0	RV=-1	RV=-1
BPX1MSS	RV=-1	RV=-1	RV=-1	RV=0	RV=0
BPX1PAS	RV=0	RV=-1	RV=-1	RV=0	RV=0
BPX1SEL	RV=0	RV=-1	RV=-1	RV=0	RV=0
BPX1SIA	RV=0	RV=-1	RV=0	RV=-1	RV=-1
BPX1SIP	RV=0	RV=-1	RV=0	RV=0	RV=0
BPX1SLP	RV=Seconds	RV=Abend	RV=Abend	RV=Seconds	RV=Seconds
BPX1SPB	RV=0	N/A	N/A	N/A	N/A
BPX1SPM	RV=0	RV=-1	RV=0	RV=0	RV=0
BPX1SSU	RV=0	RV=-1	RV=-1	RV=0	RV=0

#### Notes:

##### PRB

Program request blocks are created by MVS system services such as LINK. PRBs are also created for ESTAE routines.

**RV**

Return value returned in the service.

**N/A**

Not applicable

## Signal delivery keys

---

Signal delivery also depends on the signal delivery key. Each process has one signal delivery key. The signal delivery key is set to the PSW key of the caller of the first z/OS UNIX call that created the process. A process created by the fork or exec service has key 8. The attach\_exec service works differently from the exec and fork service; it creates a process with a signal delivery key equal to the PSW key of the Attach\_exec caller. Key zero is not a valid signal delivery key. Therefore, if the caller's PSW key is zero when mvssigsetup created the process, the mvssigsetup call fails and signal catchers cannot be invoked in this process.

## Delayed signal delivery

---

Asynchronous signals are generated from a process or task different from the task the signal is being delivered to. Delivery of asynchronous signals is not always possible and can have a delay. Signals that must be delayed are delivered later, when signals are permitted and the next z/OS UNIX service is invoked. The following describes some additional cases when signal delivery must be delayed:

- During STORAGE obtains or releases that use a hardware linkage stack.
- During execution of system services that are entered with PC or that use the hardware linkage stack (such as a BAKR instruction).
- When applications use a BAKR instruction on entry to save registers in a hardware linkage stack and use a PR instruction to restore registers on exit. Therefore, asynchronous signals cannot be delivered after the BAKR instruction and before the PR instruction.
- When a task that is set up for signals by a mvssigsetup service is followed by a system service call (for example, LINK) that creates another program request block (PRB).

z/OS UNIX System Services provides a signal deferral capability that allows an application to defer the receipt of signals until it is ready to accept them. You could use it, for instance, to shield an application from signal interruption during a time of critical processing. Once the section of critical code had finished, the application could receive any signals that had been deferred.

To use the signal deferral capability, the application sets the ThliDeferSignals bit on in the THLI data structure. When it is interested in receiving signals again, it sets this bit off. To see if any signals are pending, the application can check the OtcSigPending or the ThliSigPending bit. If OtcSigPending or ThliSigPending is set on, it can set ThliDeferSignals = OFF, and call BPX1GPI to drive signal delivery.

To access the THLI bit, traverse the data structures TCB, STCB, OTCB, and THLI. If the STCBOTCB (the field in the STCB that points to the OTCB) is 0, the process is not dubbed and the THLI has not been created. (However, since a process that has not been dubbed cannot receive signals, it is not necessary to set the THLI bit to defer their handling.) If there is an OTCB, the OTCBTHLI points to the THLI. Set the ThliDeferSignals bit accordingly.

For example:

```

If (stcbotcb ^= 0) then                /* Make sure the process is dubbed, the otc pointer */
                                        /* will not be zero. */
                                        */
    otcbthli->thlidefersignals = ON;    /* The otcbthli field points to the thli; set the thli */
                                        /* to defer signals. */
                                        */
...start of important stuff
                                        /* Remember not to issue any syscalls during this */
                                        /* segment of code. A syscall will force a delivery */
                                        /* of any pending signal. */
                                        */
...end of important stuff
otcbthli->thlidefersignals = off;        /* Reset the bit. */
                                        */

```

```

If otcbthli-thlisigpending = on      /* Check to see if any signals were made pending      */
                                     /* during the critical code interval.          */

then call bpx1gpi(...)              /* Make any syscall. It will have all pending signals */
                                     /* delivered.                                           */

```

This mechanism is not intended to be used by an application that is requesting z/OS UNIX system services. If a syscall is requested, any pending signals are delivered. The THLI bit is intended to shield the application from unwanted interruptions only when no syscalls are being performed.

## When signals cannot be delivered

Compilers and applications that enter states when signals cannot be delivered should invoke z/OS UNIX callable services after returning to a state where signal delivery is possible. This action ensures prompt delivery of signals. For example, a program may invoke a STORAGE obtain and getpid service. After returning from the getpid service, z/OS UNIX delivers any asynchronous signals that were generated during the STORAGE obtain.

When the SIR is unable to deliver a signal to a signal catcher routine for environmental reasons, the queue\_interrupt service is invoked from a signal interface routine (SIR). The queue\_interrupt service also delays signal delivery until the next z/OS UNIX callable service. z/OS UNIX callable services should be performed shortly after a queue\_interrupt call to ensure prompt signal delivery.

## Signals and multiple tasks created by ATTACH

This section describes processes that have multiple dubbed tasks created by using the ATTACH system service. It describes how the first dubbed task in a process can be created and how to create additional dubbed tasks using ATTACH. It also describes how signals work in a process with multiple dubbed tasks created by ATTACH.

The first dubbed thread in a process can be created with the fork callable service or the exec or execmvs callable service, or by the first call to z/OS UNIX callable service from any task in the address space. Subsequent tasks can be created in the process with the ATTACH system service. Once a program running on behalf of the task calls a z/OS UNIX callable service, the task becomes dubbed. Every dubbed task is assigned an 8-character thread ID.

The mvssigsetup and sigaction services allow only one thread in a process to set up a signal interface routine (SIR) and signal catchers. When a process contains two tasks with signals unblocked, the signal is delivered to the task that called mvssigsetup.

If signal action on delivery of a signal specifies termination, stop, or continue, the entire process is terminated, stopped, or continued. Delivery of a signal for default signal action occurs for any of the following conditions:

1. None of the threads is set up for signals by mvssigsetup and one or more threads do not have the signal blocked.
2. One of the threads is set up for signals by mvssigsetup and the signal is not blocked by the thread that called mvssigsetup.

## Signals and multiple tasks created by pthread\_create

The pthread\_create service creates dubbed tasks within the process. This section describes how signals work in processes that have multiple dubbed tasks created by the pthread\_create service and ATTACH system service.

A thread created by pthread\_create also inherits any signal setup information created by a prior mvssigsetup call. If the caller of pthread\_create had previously called mvssigsetup successfully, the thread created is also set up for signals. The mvssigsetup and pthread\_create services can be used to create multiple threads in a process that is set up for signals.

When a signal is generated by a kill service request to a process that has multiple threads set up for signals and threads that are not set up for signals, z/OS UNIX signal processing must determine which

thread has the most interest in the signal. The signal is delivered to the thread with the most interest when a signal catcher is defined by a `sigaction` call.

The following is a list of signal interest rules for a signal generated by a `kill` call from most to least interested:

1. When threads are found in a `sigwait` for this signal, the signal is delivered to the first thread found in a `sigwait`.
2. When all threads are blocking this signal, the signal is left pending at the process level. The `sigpending` service moves blocked pending signals at the process level to the thread-level.
3. When the default terminating signal action (not ignore and not catch) is to take place, that action is performed for all threads in the process.
4. When all of the following are true:
  - One or more threads are set up for signals.
  - All threads set up for signals have the signal blocked.
  - A thread not set up for signals has not blocked the signal.

The signal is left pending on the first thread set up for signals. This signal remains pending on that thread until the thread unblocks the signal.

5. When one or more threads are set up for signals and at least one of the threads set up for signals has the signal unblocked, the signal is delivered to the first thread that is set up for signals that also has the signal unblocked.

## Signal defaults

This section contains information about the signals that are supported by z/OS UNIX. These signals are mapped by the `BPXYSIGH` mapping macro; see “`BPXYSIGH` — Signal constants” on page 1148. The following table lists the signals and their default actions:

Constant	Value	Default action	Description
<b>SIGABND#</b>	18	1	Abend
<b>SIGABRT#</b>	3	1	Abnormal termination
<b>SIGALRM#</b>	14	1	Timeout
<b>SIGBUS#</b>	10	1	Bus error
<b>SIGCHLD#</b>	20	2	Child process terminated or stopped
<b>SIGCONT#</b>	19	4	Continue if stopped
<b>SIGDANGER</b>	33	1	Termination
<b>SIGDUMP#</b>	39	2	The system takes a <code>SYSMDUMP</code> and writes it to an MVS data set or a z/OS UNIX file. The <code>_BPXK_MDUMP</code> environment variable must be set to the name of the data set or file. This signal cannot be caught.
<b>SIGFPE#</b>	8	1	Erroneous arithmetic operation, such as division by zero or an operation resulting in overflow
<b>SIGHUP#</b>	1	1	Hangup detected on controlling terminal
<b>SIGILL#</b>	4	1	Detection of an incorrect hardware instruction
<b>SIGINT#</b>	2	1	Interactive attention
<b>SIGIO#</b>	23	2	Completion of input or output



Constant	Value	Default action	Description
<b>SIGIOER#</b>	27	2	I/O error
<b>SIGKILL#</b>	9	1	Termination (cannot be caught or ignored). Can result if abend not caught or handled and terminating status not set; CPU time exceeded and SIGXCPU# caught or ignored; or <code>sigkill</code> shell command sent.
<b>SIGNULL#</b>	0	2	Null; no signal sent (cannot be caught or ignored)
<b>SIGPIPE#</b>	13	1	Write on a pipe with no readers
<b>SIGPOLL#</b>	5	1	Pollable event
<b>SIGPROF#</b>	32	1	Profiling timer expired
<b>SIGQUIT#</b>	24	1	Interactive termination
<b>SIGSEGV#</b>	11	1	Detection of an incorrect memory reference
<b>SIGSTOP#</b>	7	3	Stop (cannot be caught or ignored)
<b>SIGSYS#</b>	12	1	Bad system call
<b>SIGTERM#</b>	15	1	Termination
<b>SIGTHCONT#</b>	35	1	Thread continue (cannot be caught or blocked or ignored)
<b>SIGTHSTOP#</b>	34	1	Thread stop (cannot be caught or blocked or ignored)
<b>SIGTMOUT#</b>	40	1	Terminates a process waiting for terminal activity. Cannot be caught or ignored. Is reserved for z/OS use. It cannot be sent by an application program.
<b>SIGTRACE#</b>	37	2	Toggles the user syscall trace setting ON or OFF.
<b>SIGTSTP#</b>	25	3	Interactive stop
<b>SIGTTIN#</b>	21	3	Read from a control terminal attempted by a member of a background
<b>SIGTTOU#</b>	22	3	Write from a control terminal attempted by a member of a background process group
<b>SIGTRAP#</b>	26	1	Trap used by the <code>ptrace</code> call
<b>SIGURG#</b>	6	2	High bandwidth data is available at a socket
<b>SIGUSR1#</b>	16	1	Reserved as application-defined signal 1
<b>SIGUSR2#</b>	17	1	Reserved as application-defined signal 2 process group
<b>SIGVTALRM#</b>	31	1	Virtual timer expired
<b>SIGXCPU#</b>	29	1	CPU time limit exceeded
<b>SIGXFSZ#</b>	30	1	File size limit exceeded
<b>SIGWINCH#</b>	28	2	Change size of window

The default actions are:

1. Abnormal termination.

2. Ignore the signal.
3. Stop the process.
4. Continue if it is currently stopped; otherwise, ignore the signal.

---

## Appendix H. Using threads with callable services

z/OS UNIX threads are tasks that are using z/OS UNIX services. Pthreads are z/OS UNIX threads that are created with `pthread_create`; this also includes the initial pthread-creating task (IPT). The first thread in a process to invoke the `pthread_create` service becomes the IPT. This topic contains information about creating pthreads, the IPT, terminating pthreads, and multiple pthreads. It also shows scenarios for different termination situations.

---

### Creating threads

Threads are created as follows.

The successful completion of:

- The `pthread_create` service
- The fork or exec service
- Most z/OS UNIX service requests from an undubbed MVS task

A single-threaded process is created with fork, with exec, or by the invocation of a kernel service from within an MVS address space.

Multiple-threaded processes can be created with `pthread_create`.

The IPT and all `pthread_create` threads are referred to as *pthreads*. All future `pthread_create` requests attach subtasks to the IPT, even though they are not issued by the IPT. This is important in thread termination. For a complete description of the process of creating threads, see [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread”](#) on page 581.

### The `pthread_create` task initialization routine

The first routine that is given control in the new task when a thread is created with the `pthread_create` service is the `pthread_create` pthread-creating task initialization routine. (The pthread-creating task initialization routine is not the same as the initial pthread-creating task (IPT). The pthread-creating task initialization routine is the routine that is given control when a `pthread_create` is done, whereas the IPT refers to the task that the first task runs on.) The `pthread_create` pthread-creating task initialization routine does the following:

1. Acquires task-related resources required by the user application.
2. Calls `pthread_exit_and_get` service to exit the old thread and get the new thread information. The exit of the old thread is ignored if this is the first call to `pthread_exit_and_get`.
3. Checks for failures. If a failure is found, it skips to step “8” on page 1421.
4. Gets pthread-related resources for the newly created thread.
5. Calls the user-specified `Start_routine`.
6. Releases resources for the newly created thread.
7. Repeats step “2” on page 1421.
8. Releases task-related resources.
9. Returns to the caller (ends the task).

**Note:** When control is returned after a successful `pthread_exit_and_get` call, the thread can be interrupted by any signals that are not blocked. The signal blocking mask of the created thread is inherited from the thread that invoked the created thread.

## Terminating pthreads

**Note:** If multiple threads are created with a combination of `pthread_create` and dubbed MVS tasks, the following termination methods do not apply. The exception to this, of course, is that the IPT is a dubbed task. If the IPT has any subtasks that are non-pthread threads, the following termination scenarios also do not apply.

There are no prescribed methods for terminating threads that are mixed with other dubbed tasks in a single process.

There are three ways to terminate a thread without exiting the process:

- The `pthread_exit_and_get` (BPX1PTX) service terminates the thread that invoked it. If it is successful, control is returned to the invoking task.
- The `pthread_cancel` (BPX1PTB) service generates a cancel request to the target thread. After the cancel request is delivered, the thread and its associated task are terminated by the kernel. This behavior can be circumvented if the thread intercepts the cancelation request (see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals”](#) on page 487).
- The `pthread_quiesce` (BPX1PTQ) service sends a quiesce event to all other pthreads in the process. If the other pthreads do not intercept the quiesce event (see [“mvssigsetup \(BPX1MSS, BPX4MSS\) — Set up MVS signals”](#) on page 487), delivery of the event terminates the thread and the task, if the target is not the IPT.

The two types of threads that are created with `pthread_create` require different actions for terminating.

### Heavyweight thread (HWT)

Terminating a heavyweight thread (HWT) also requires terminating its associated task. After the `pthread_exit_and_get` service is issued to exit the HWT, z/OS UNIX services can no longer be issued from that task. The only exception is the `mvsprocclp` service (BPX4MPC), which remains available.

### Mediumweight thread (MWT)

Terminating an MWT does not require that the task terminate. You can terminate it by using the `PTGETNEWTTHREAD` option on the `pthread_exit_and_get` service. The pthread-creating task initialization routine that is specified on the `pthread_create` service can repeatedly call the `pthread_exit_and_get` service, getting new thread requests as they are created. This avoids the overhead of task creation and termination for each thread.

For information on HWTs and MWTs, see [“pthread\\_create \(BPX1PTC, BPX4PTC\) — Create a thread”](#) on page 581.

### Terminating multiple pthreads and tasks

Terminating a pthread is different from terminating the task that the pthread runs on. The IPT should be the last task to terminate; that is, the IPT should wait for all pthreads and supporting tasks to terminate before it terminates. If the IPT and its associated task should terminate before all its subtasks terminate, those subtasks abend asynchronously with a 33E abend. This type of termination does not allow an orderly cleanup of pthread and task-related resources.

When a process contains multiple pthreads, and one of the executing pthreads starts process termination, the following steps should be taken:

1. The terminating pthread uses the `pthread_quiesce` (BPX1PTQ) service to inform all other pthreads that are running in the process of its process termination.
2. The `pthread_quiesce` service places the issuing pthread in a wait state until all other pthreads are notified and have terminated.
3. As each pthread's signal interface routine receives the quiesce notification, it uses the `pthread_exit_and_get` service to terminate the pthread. The signal interface routine should not

pass control to the user program, because it might continue processing. The task that invoked `pthread_quiesce` is waiting for all the pthreads in the process to terminate.

4. The pthread that is issuing the `pthread_quiesce` service gains control after all pthreads have terminated. The terminating pthread can then invoke any exit and cleanup functions that are necessary for an orderly termination of the process.

**Note:** The tasks that supported quiesced pthreads can still be running after control is returned to the task that issued `pthread_quiesce`. Only the pthreads have terminated, not the tasks. Terminating the task is a separate and asynchronous part of terminating the process.

5. The terminating pthread can then issue a terminating service request such as `exit`, `_exit`, or `exec`. If the terminating pthread is the IPT, the `mvspocclp` (BPX1MPC) service can be issued instead of the `_exit` (BPX1EXI) service. This avoids the automatic termination of the task.
6. The IPT gains control only when all the pthreads that were created with `pthread_create` have terminated. The IPT can then call `mvspocclp` (BPX1MPC) to clean up the remaining z/OS UNIX environment. Control cannot return to the IPT until all the other tasks that supported the pthreads have exited. If any of the pthread subtasks fail to terminate, `mvspocclp` sets a failing return code.
7. Now that all the tasks have terminated (except for the IPT), control is returned to the caller of the application (if one exists) or back to the system (which terminates the IPT).

## Pthread termination scenarios

The following scenarios describe the steps needed to terminate multithread processing for situations application programmers might encounter.

### Using `exit` or `_exit` when the thread is not the IPT

Table 38 on page 1423 describes the actions that are taken for `exit` or `_exit` issued from a thread created with `pthread_create`.

Table 38. Using <code>exit</code> or <code>_exit</code> when the thread is not the IPT		
Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1	Not applicable	A request to exit the process was issued.
2	Not applicable	A <code>pthread_quiesce</code> is issued. Control is not returned until all other pthreads in this process end with <code>pthread_exit_and_get</code> .
3	An asynchronous quiesce event is delivered to this thread.	Not applicable
4	Either the thread is terminated by the kernel, or the signal interface routine intercepts the quiesce termination event to do necessary thread cleanup and issue another <code>pthread_exit_and_get</code> . Interception of quiesce events must be specified by the <code>mvssigsetup</code> service.	Not applicable
5	The IPT thread is terminated, and the IPT is placed in a wait state in the kernel.	Not applicable
6	Not applicable	Control is returned from <code>pthread_quiesce</code> when all other pthreads terminate.

<i>Table 38. Using exit or _exit when the thread is not the IPT (continued)</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
<b>8</b>	Not applicable	An _exit service request is issued to terminate the process and pass the process status. This pthread and task are both terminated, and control is not returned to the _exit service caller.
<b>9</b>	The kernel posts the IPT when the last pthread terminates.	Not applicable
<b>10</b>	The mvspocclp service is issued to clean up any remaining portions of the process. Control returns from this service after all subtasks created with pthread_create terminate, or when a reasonable amount of time to do this has elapsed.	Not applicable
<b>11</b>	The IPT gains control after the mvspocclp service completes. All pthreads for this process and all subtasks of the IPT have terminated.	Not applicable
<b>12</b>	The IPT is no longer associated with the kernel and can now return to its caller or to the system.	Not applicable

### Using exit or \_exit when the thread is the IPT

Table 39 on page 1424 describes the actions that are taken for exit or \_exit issued from the IPT thread.

<i>Table 39. Using exit or _exit when the thread is the IPT</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
<b>1</b>	A request to exit the process is issued from the IPT.	Not applicable
<b>2</b>	A pthread_quiesce is issued. Control is not returned until all other pthreads in this process end with pthread_exit_and_get.	Not applicable
<b>3</b>	Not applicable	An asynchronous quiesce event is delivered to this thread.
<b>4</b>	Not applicable	Either the thread and its associated task are terminated by the kernel, or the signal interface routine intercepts the quiesce termination event to do necessary thread cleanup and to issue another pthread_exit_and_get. Interception of quiesce events is specified by the mvssigsetup service.
<b>5</b>	Not applicable	Control is returned to the pthread-creating task initialization routine (QUIESCE_TERM only), the remaining parts of the environment are cleaned up, and control is returned to the caller, terminating the task.

<i>Table 39. Using exit or _exit when the thread is the IPT (continued)</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
<b>6</b>	Control is returned after the pthread_quiesce call when all other pthreads terminate (perhaps not all tasks have terminated yet).	Not applicable
<b>7</b>	Process the remaining thread and clean up (such as running exits).	Not applicable
<b>8</b>	Terminate the process and pass the process status with mvspocclp status.	Not applicable
<b>9</b>	When control is returned from mvspocclp, all pthreads for this process and all subtasks of the IPT have terminated.	Not applicable
<b>10</b>	The IPT task is no longer associated with the kernel and can now return to its caller or to the system.	Not applicable

## Using pthread\_exit\_and\_get when the thread is not the IPT and not the last thread

Table 40 on page 1425 describes the actions that are taken when pthread\_exit\_and\_get is issued on a thread that is not the IPT and is not the last thread.

<i>Table 40. Using pthread_exit_and_get when the thread is not the IPT and not the last thread</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
1		A request to exit the pthread is issued.
2		Run thread cleanup routines before terminating this thread.
3		Return to the pthread-creating task initialization routine that issued pthread_exit_and_get to terminate the thread, using the PTEXITTHREAD and PTGETNEWTHREAD option for MWTs or the PTEXITTHREAD option for HWTs. If you want to know when the last thread is terminating so that process termination cleanup can be done first, specify PTFAILIFLASTTHREAD. You must then call pthread_exit_and_get again, but this time without the PTFAILIFLASTTHREAD option.
4		For MWTs, this task waits in the kernel until the next new pthread_create request. When pthread_exit_and_get returns a -1 return value, a new thread was not created. You must exit the pthread-creating task initialization routine, terminating the task. (You must always do this for HWTs.)

Table 40. Using pthread\_exit\_and\_get when the thread is not the IPT and not the last thread (continued)

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
5		A successful return from pthread_exit_and_get indicates that this was not the last thread that terminated. If the PTEXITTHREAD and PTGETNEWTHREAD option was used, a new thread was returned.

## Using pthread\_cancel when the thread is not the last thread and is canceled

Table 41 on page 1426 defines the actions that are taken when the pthread\_cancel request is handled by the signal interface routine, and the cancel causes the thread to terminate. This is the same as when the target thread issues pthread\_exit\_and\_get. The status of the thread is -1, and is available for joining threads.

Table 41. Using pthread\_cancel when the thread is not the last thread and is canceled

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1		The pthread_cancel request was received and delivered to the signal interface routine. Interception of cancelations must be specified by the mvssigsetup service.
2		Set Status_field in the pthread_exit_and_get service to -1. See “pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread” on page 588.
3		Now follow the steps in Table 40 on page 1425.

## Using pthread\_exit\_and\_get when the thread is the IPT and not the last thread

Table 42 on page 1426 describes the actions that are taken when pthread\_exit\_and\_get is issued on a thread that is the IPT and is not the last thread. The IPT is placed in wait state until all other pthreads in this process terminate.

Table 42. Using pthread\_exit\_and\_get when the thread is the IPT and not the last thread

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1	A request to exit the pthread was issued.	
2	Run thread cleanup routines before terminating this thread.	
3	To terminate the thread, issue the pthread_exit_and_get service with the PTEXITTHREAD option. To determine when the last thread has terminated so that process termination cleanup can be done first, use the pthread_exit_and_get service with the PTFAILIFLASTTHREAD option. Then repeat the pthread_exit_and_get service, but without the PTFAILIFLASTTHREAD option.	



<i>Table 42. Using pthread_exit_and_get when the thread is the IPT and not the last thread (continued)</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
<b>4</b>	The IPT is now in a wait state until the process terminates.	
<b>5</b>	A return from pthread_exit_and_get indicates that all other pthreads for the process have terminated.	
<b>6</b>	The mvspocclp service is issued to clean up any remaining portions of the process. Control returns from this call after all subtasks that were created with pthread_create terminate, or until time to do so has elapsed.	
<b>7</b>	The IPT task gains control after the mvspocclp call. All pthreads for this process and all subtasks of the IPT have terminated.	
<b>8</b>	The IPT task is no longer associated to the kernel, and can now return to the caller or to the system.	

## Using pthread\_exit\_and\_get when the thread is not the IPT and is the last thread

Table 43 on page 1427 describes the actions that are taken when pthread\_exit\_and\_get is issued on a thread that is not the IPT and is the last thread.

<i>Table 43. Using pthread_exit_and_get when the thread is not the IPT and is the last thread</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
<b>1</b>	The IPT is in a wait state because of a previous pthread_exit_and_get.	pthread_exit_and_get is issued from this thread.
<b>2</b>		Run thread cleanup routines before this thread terminates.
<b>3</b>		Return to pthread-creating task initialization routine that issues pthread_exit_and_get to exit the thread, using the PTEXTTHREAD and PTGETNEWTHREAD option for MWTs or the PTEXTTHREAD option for HWTs. If you want to know when the last thread is terminating so that process termination cleanup can be done first, specify the PTFAILIFLASTTHREAD option. You must then call pthread_exit_and_get again, but this time without the PTFAILIFLASTTHREAD option.
<b>4</b>		A failing return value and reason code from pthread_exit_and_get indicates that this is the last thread.
<b>5</b>		Process the remaining thread and clean up (such as running exits).

<i>Table 43. Using pthread_exit_and_get when the thread is not the IPT and is the last thread (continued)</i>		
<b>Step</b>	<b>Thread 1 (initial pthread-creating task, or IPT)</b>	<b>Thread 2 (pthread-created thread)</b>
<b>6</b>		Call pthread_exit_and_get without the PTFAILIFLASTTHREAD option to terminate the last thread and the process.
<b>7</b>		Clean up any MVS resources that may have been obtained STAE/SPIE/storage, after control is returned from pthread_exit_and_get to the pthread-creating task initialization routine.
<b>8</b>		The pthread-creating task initialization routine returns to its caller, terminating the task. The IPT is posted when this task terminates.
<b>9</b>	The IPT gains control after its pthread_exit_and_get and all threads have terminated.	
<b>10</b>	Issue the BPX1MPC service to clean up any remaining portions of the process. Control returns from this call after all subtasks created with pthread_create terminate, or until the time to do so has elapsed.	
<b>11</b>	The IPT task gains control when control is returned from mvspoclp and all pthreads for this process and all subtasks of the IPT have terminated.	
<b>12</b>	The IPT task is no longer associated with the kernel, and can now return to its caller or to the system.	

### Using pthread\_exit\_and\_get when the IPT is the last thread

Table 44 on page 1428 describes the actions that are taken when pthread\_exit\_and\_get is issued for a thread that is the IPT and is the last thread.

<i>Table 44. Using pthread_exit_and_get when the IPT is the last thread</i>		
<b>Step</b>	<b>IPT task is the only task</b>	<b>Thread 2 doesn't exist</b>
<b>1</b>	A request to exit the pthread was issued.	
<b>2</b>	Run thread cleanup routines before this thread terminates.	
<b>3</b>	Call pthread_exit_and_get with the PTEXITTHREAD and PTFAILIFLASTTHREAD options to terminate the thread on the IPT.	
<b>4</b>	A return value and reason code reporting a failure from pthread_exit_and_get indicates that this is the last thread.	
<b>5</b>	Process the remaining thread and cleanup (such as running exits).	

<i>Table 44. Using pthread_exit_and_get when the IPT is the last thread (continued)</i>		
<b>Step</b>	<b>IPT task is the only task</b>	<b>Thread 2 doesn't exist</b>
<b>6</b>	Call pthread_exit_and_get without the PTFAILIFLASTTHREAD option to terminate the last thread and the process.	
<b>7</b>	Control is returned to the IPT from pthread_exit_and_get, and all threads terminate.	
<b>8</b>	The mvspocclp service is issued to clean up any remaining portions of the process. Control returns from this service after all subtasks created with pthread_create terminate, or until the time to do so has elapsed.	
<b>9</b>	The IPT task gains control when the mvspocclp service completes. All pthreads for this process and all subtasks of the IPT have terminated.	
<b>10</b>	The IPT task is no longer associated with the kernel, and can now return to its caller or to the system.	



## Appendix I. Optimizing performance using process- and thread-level information

The process-level information area (PRLI) and the thread-level information area (THLI) contain information that can be used to optimize the performance of certain callable services. This information describes how to access the information in these areas and how the information can be used.

A thread-level information area (THLI) is created for each task in the system. The THLI is pointed to by the OTCB field OTCBTHLI. The OTCB is pointed to by a secondary task control block field, STCBOTCB.

A process-level information area (PRLI) is created for each process in the system. The PRLI is pointed to by the THLI field THLIPRLI for each task in the process.

The system maintains information in the PRLI and THLI that can be used to reduce the system overhead that is associated with certain callable services and improve their performance. The callable services that can use the information in these control blocks include:

- BPX1PSI, BPX4PSI (pthread\_setintr)
- BPX1PST, BPX4PST (pthread\_setintrtype)
- BPX1SPM, BPX4SPM (sigprocmask)
- BPX1GPI, BPX4GPI (getpid)

### Optimization processing for BPX1PSI, BPX4PSI (pthread\_setintr)

Information in the THLI area can be used to optimize pthread\_setintr (BPX1PSI, BPX4PSI) callable service invocations. BPX1PSI (BPX4PSI) must not be optimized if a signal is pending for the thread. A signal pending condition is indicated by the ThliSigPending flag. When this flag is on, indicating that a signal is pending, BPX1PSI (BPX4PSI) must be called to process the request and process signal delivery.

Table 45 on page 1431 maps the actions that can be taken for BPX1PSI (BPX4PSI) when there is no signal pending. The result column shows the action that the optimizing program can take. The cancel pending column reflects the setting of ThliCancelPending, and the current state column that of ThliCancelDisabled. The new state is provided by the caller of BPX1PSI (BPX4PSI). The interruptability type, which is set by BPX1PST (BPX4PST), is not applicable to BPX1PSI (BPX4PSI) processing.

Table 45. Optimization processing for BPX1PSI, BPX4PSI (pthread_setintr)				
The current state	New state	Interruptability type	Cancel pending	Result
<b>Disabled</b>	Disabled	Not applicable	Not applicable	Return "disabled"
<b>Enabled</b>	Disabled	Not applicable	Not applicable	Issue BPX1PSI (BPX4PSI)
<b>Enabled</b>	Enabled	Not applicable	Yes	Issue BPX1PSI (BPX4PSI)
<b>Enabled</b>	Enabled	Not applicable	No	Return "enabled"
<b>Disabled</b>	Enabled	Not applicable	Not applicable	Issue BPX1PSI (BPX4PSI)
<b>Not applicable</b>	Invalid	Not applicable	Not applicable	Issue BPX1PSI (BPX4PSI)

In other words, the optimizing program should issue the BPX1PSI (BPX4PSI) if there is a request to change the interruptability state, or if the state is enabled and there is a cancel pending, as indicated by the ThliCancelPending bit.

## Optimization processing for BPX1PST, BPX4PST (pthread\_setintrtype)

Information in the THLI area can be used to optimize pthread\_setintrtype (BPX1PST, BPX4PST) callable service invocations.

Table 46 on page 1432 maps the actions that can be taken for BPX1PST (BPX4PST) when there is no signal pending. The result column shows the action that the optimizing program can take. The cancel pending column reflects the setting of ThliCancelPending, the cancel disabled column the setting of ThliCancelDisabled, and the current interruptability type column the setting of ThliCancelAsync. The new interruptability type is provided by the caller of BPX1PST (BPX4PST).

Table 46. Optimization processing for BPX1PST, BPX4PST (pthread_setintrtype)				
The current int. type	New interruptability type	Cancel disabled	Cancel pending	Result
<b>Controlled</b>	Controlled	Not applicable	Not applicable	Return "controlled"
<b>Asynch</b>	Controlled	Not applicable	Not applicable	Issue BPX1PST
<b>Controlled</b>	Asynch	Not applicable	Not applicable	Issue BPX1PST (BPX4PST)
<b>Asynch</b>	Asynch	Yes	Not applicable	Return "asynch"
<b>Asynch</b>	Asynch	No	No	Return "asynch"
<b>Asynch</b>	Asynch	No	Yes	Issue BPX1PST (BPX4PST)
<b>Not applicable</b>	Invalid	Not applicable	Not applicable	Issue BPX1PST (BPX4PST)

In other words, the optimizing program should issue the BPX1PST (BPX4PST) if there is a request to change the interruptability type; or if the type is asynchronous and cancel is not disabled (ThliCancelDisabled off) and there is a cancel pending (ThliCancelPending on).

## Optimization processing for BPX1SPM, BPX4SPM (sigprocmask)

Information in the THLI data area can be used to optimize sigprocmask (BPX1SPM, BPX4SPM) invocations.

The optimizing program should first process the new mask that is provided by the caller of BPX1SPM (BPX4SPM), to determine if optimization is possible. If no new mask is provided, no change is being made to the signal mask, and this call can be optimized.

To process the new mask, the optimizing program should first generate the effective new mask using the new mask provided by the caller, clearing bits from this new mask for any signals that cannot be caught (ANDing the provided mask with PrliCatcherMask), and then applying the How requested by the caller as shown in Table 47 on page 1433.

Table 47. Optimization processing for BPX1SPM, BPX4SPM (sigprocmask)

How	Effective mask
<b>SIG_SETMASK</b>	New mask ANDed with PrliCatcherMask
<b>SIG_BLOCK</b>	New mask ORed with ThliSigMask
<b>SIG_UNBLOCK</b>	Complement new mask ANDed with ThliSigMask
<b>Other</b>	An incorrect How was specified; issue BPX1SPM (BPX4SPM) or fail the request with an appropriate error code.

If the effective mask does not equal the current mask in ThliSigMask, a change in value of the current signal mask must be made, and BPX1SPM (BPX4SPM) should be issued. If the effective mask is the same as the current signal mask, the request is a NOOP and may be optimized.

If the request is being optimized and the caller requested that the previous value of the signal mask be returned, the optimizing program should return ThliSigMask to the caller.

## Optimization processing for BPX1GPI, BPX4GPI (getpid)

Information in the PRLI data area can be used to optimize the getpid callable service invocations.

The optimizing program should return PrliProcessID if there is no signal pending; otherwise, getpid should be issued.





## Appendix J. Process start and end exits for z/OS UNIX

Four installation exits are defined to enable applications to monitor z/OS UNIX process activity. Exit routines can be added to each exit point. z/OS UNIX passes control to the exit routine when an exit point is reached, and information about the current process and its creator is then passed to the exit routine. These are the installation exits:

### Preprocess initiation exit (BPX\_PREPROC\_INIT)

Preprocess initiation exit routines receive control immediately before the creation of any new z/OS UNIX process. When a preprocess initiation exit routine receives control, the Process Exit Data Block (PEDB) contains the data about the initiating job.

Upon return from the exit, if the exit's return code is greater than 4, the process initiation request is rejected. The z/OS UNIX callable service that drove this process initiation request will fail with a return value of -1, a return code of EAGAIN, and a reason code of JrPreProcInitExitReject.

The preprocess initiation exit should have a recovery routine to clean up any resources that it obtained. If the exit does not have a recovery routine, first-failure capture is not possible, and resources that were obtained will not be released. Should an exit abend, the z/OS UNIX callable service that drove this process initiation request will fail with a return value of -1, a return code of EAGAIN, and a reason code of JrPreProcInitExitAbend.

The sole purpose of the preprocess initiation exit point is to provide an application with the ability to fail an attempt to initialize a process. If this is not the intent of your exit routine, you should not use this exit point. Do not use this exit point if, for example, your primary purpose is to monitor the initialization and termination of processes in the system, because it does not receive enough information to identify the process that is to be initialized. When it receives control, the only information the exit has available (from the PEDB) is the unique ID and information about the initiator of the process.

**Tip:** Resources should not be obtained at this exit point, because it is possible that another exit routine could subsequently fail the process initialization attempt, and no further exit points would be driven for this process, including the process termination exit. Resources that relate to the process should be obtained in the post-process initialization exit, where the process is fully initialized, and the termination exit will eventually run upon termination of the process.

**Restriction:** This exit should not use any z/OS UNIX callable service. To do so could cause unexpected results, such as ABEND 138–ENQ deadlock.

### Postprocess initiation exit (BPX\_POSPROC\_INIT)

Postprocess initiation exit routines receive control immediately after the creation of any new z/OS UNIX process. When a post-process initiation exit routine receives control, the process exit data block (PEDB) contains the creator and the new process data.

The postprocess initiation exit should have a recovery routine to clean up any resources that it obtained. If the exit does not have a recovery routine, first-failure capture is not possible, and resources that were obtained will not be released. Should an exit abend, the z/OS UNIX callable service that drove this process initiation request will fail with a return value of -1, a return code of EAGAIN, and a reason code of JrPosProcInitExitAbend.

**Rule:** This exit should not use any z/OS UNIX callable service. To do so could cause unexpected results, such as ABEND 138–ENQ deadlock.

### Process image initiation exit (BPX\_IMAGE\_INIT)

Process image initiation exit routines receive control immediately before the initiation of a new z/OS UNIX process image. This occurs when a successful spawn, attach\_exec, attach\_execmvs, exec, or execmvs callable service is done. The process image initiation exit receives control before the new

process image file is run. When a process image initiation exit routine receives control, the process exit data block (PEDB) contains the data of the creator and the new image.

The process image initiation exit should have a recovery routine to clean up any resources it obtained. If the exit does not have a recovery routine, first-failure capture is not possible, and resources that were obtained will not be released. If an exit abends, the z/OS UNIX callable service that drove this process receives a successful return code, but the image is not created, and an EC6 ABEND with a ImageInitExitABEND reason code is issued.

**Rule:** This exit should not use any z/OS UNIX callable service. To do so could cause unexpected results, such as ABEND 138-ENQ deadlock.

#### Preprocess termination exit (BPX\_PREPROC\_TERM)

Preprocess termination exit routines receive control immediately before the termination of a z/OS UNIX process. These exits might receive control in the address space of the process or in the master address space, if the address space of the process was terminated. In the latter case (ASID=1), z/OS UNIX callable services cannot be used by the exit. When a preprocess termination exit receives control, the process exit data block (PEDB) contains data about the terminating process.

## Exit environment

The user exit receives control in the following environment:

- Supervisor state, key zero.
- Running in the ASID of the process, except for the pre-process termination exit, which runs in the master address space if the address space of the process was terminated.

Register usage:

- On entry to the user exit, register 1 points to the Process Exit Data Block (PEDB).
- For the preprocess initiation exit, if the value returned in register 15 is > 4, the process initiation request is rejected. For all other exit points, the return code in register 15 is ignored.

## Errno and errnoJrs

Any callable service that causes a process to be dubbed can receive the following errno and errnojr combinations:

Error	Description
<b>Return code EAGAIN, reason code JrPreProcInitExitReject.</b>	The preprocess initiation exit failed the process initiation request.
<b>Return value -1, return code EAGAIN, reason code JrPreProcInitExitAbend</b>	The preprocess initiation exit ended abnormally.
<b>Return value -1, return code EAGAIN, reason code JrPosProcInitExitAbend</b>	The postprocess initiation exit ended abnormally.
<b>EC6 ABEND, reason code ImageInitExitABEND</b>	The process image initiation exit ended abnormally.

## Restrictions

1. Process start/end exits cannot use any z/OS UNIX callable services.
2. Exit routines are responsible for cleaning up any resources they obtain (such as storage or locks).
3. Exit routines should have recovery routines to ensure first-failure data capture.

## Usage notes

---

The same exit point can be used for all four exits. The value in the PEDB field PEDBEXITPOINTID identifies the exit point that is hit. For example, If PEDBEXITPOINTID is PEDB\_BPX\_PREPROC\_INIT, the preprocess initiation exit point is hit. The constants that identify each exit point are defined at the bottom of the PEDB (see [“BPXYPEDB — Mapping of process exit data block”](#) on page 1114).

See [Using installation exits](#) in *z/OS UNIX System Services Planning* for more information about the process start/end installation exits.



---

## Appendix K. Dynamic syscall exits for z/OS UNIX

Dynamic syscall exits for z/OS UNIX provide support for pre-syscall and post-syscall exit routines that will be invoked for select z/OS UNIX syscalls.

Two steps are required to utilize z/OS UNIX syscall exits.

1. First, enable the exit routines for select z/OS UNIX system calls. See [“Enabling syscalls for exits” on page 1439](#).
2. Then define the exit point as described in [“Defining exit points for syscalls” on page 1439](#).

---

### Enabling syscalls for exits

The SC\_EXITTABLE statement specifies the path name of a z/OS UNIX file containing a list of stub names for z/OS UNIX syscalls (such as BPX1xxx and BPX4xxx) that identify which syscalls are to be enabled for exits.

You can change the SC\_EXITTABLE setting dynamically by using the SETOMVS or SET OMVS commands. To make the change permanent, edit the BPXPRMxx member that is used for IPLs.

After the SC\_EXITTABLE setting has been enabled, z/OS UNIX will invoke the exit routines that were added to the presyscall and post-syscall exits points. Exit routines are invoked only for those syscalls that were enabled for exits. Syscalls are not enabled for exits by default (when SC\_EXITTABLE is not specified).

For the syntax of the SC\_EXITTABLE statement, see [BPXPRMxx \(z/OS UNIX System Services parameters\)](#) in [z/OS MVS Initialization and Tuning Reference](#).

---

### Defining exit points for syscalls

During kernel initialization, z/OS UNIX defines two exit points to the dynamic exits facility, BPX\_PRE\_SYSCALL and BPX\_POST\_SYSCALL. Both exit points are defined with the following attributes:

- AMODE(31)
- REENTRANT(REQ)
- PERSIST(IPL)
- ABENDNUM(5)
- ABENDCONSEC(NO)
- FASTPATH(NO)
- LOADAPF(YES)
- EXITTYPE(INSTALLATION)
- KEY(0)

The BPX\_PRE\_SYSCALL exit point, as the name implies, is for exit routines that are invoked before the syscall service is invoked. The BPX\_POST\_SYSCALL exit point is for exit routines that are invoked after the syscall service has completed.

For more information about exit point attributes, see [CSVDYNEX - Provide dynamic exit services in z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN](#).

### Managing the exit routines

Use the dynamic exits facility to manage exit routines. To manage dynamic exits, you can use the CSVDYNEX macro, the SETPROG EXIT command, and the PROG parmlib member in SYS1.PARMLIB.

## Exit routine environment

Environment at entry to exit routines:

Table 48. Environment at entry to exit routines	
Operation	Environment
Authorization	Supervisor state, PSW key zero.
Dispatchable unit mode	Task.
Cross-memory mode	PASN=HASN.
AMODE	31-bit.
ASC mode	Primary.
Address space	Caller's.

Registers at entry to exit routines:

Table 49. Registers at entry to exit routines	
Register at entry	Notes
R0	Contains no information for use by the exit routine.
R1	Address of BPXYSXIB.
R2-12	Contains no information for use by the exit routine.
R13	Address of 72-byte save area (below the bar, key 0).
R14	Return address.
R15	Address of exit routine.

**Note:** BPXYSXIB is the syscall information block. It is a data structure passed by z/OS UNIX to user-defined exit routines and is located in SYS1.MACLIB. It maps the input and output information between the z/OS UNIX kernel and the exit routines. There is one SXIB per z/OS UNIX thread. The SXIB is shared by all exit routines that are invoked when an exit point is reached. To find information about syscall numbers, see Appendix L, “Callable service numbers,” on page 1445. For more information about the format of parameter lists, see “Call parameter lists” on page 4.

Registers at return from exit routines:

Table 50. Registers at return from exit routines	
Register at return	Notes
R0-R1	Need not contain any particular value.
R2-R14	Restored to contents at entry.
R15	Return code (pre-syscall exits only). Otherwise, need not contain any particular value.

## Exit routine rejection of a syscall

Pre-syscall (BPX\_PRE\_SYSCALL) exit routines have the option to prevent a syscall service routine from running. A pre-syscall exit routine can reject a syscall by returning a nonzero return code in register 15. When an exit routine returns a nonzero return code, the z/OS UNIX kernel prevents the syscall service routine from running and sets the following values in the caller's return parameters.

Table 51. Values of nonzero return codes	
Return status	Values
Return_value	-1 (minus 1).
Return_code	EAGAIN.
Reason_code	JrFailedBySyscallExit (x'0663').

In addition to setting the return code in register 15, exit routines that reject syscalls should also store additional information in the SXIB. This information will aid in identifying the exit routine that rejected the syscall and why the syscall was rejected. The following fields are provided in the SXIB for that purpose.

- SxibPreScID (16-byte field to identify the application or exit routine).
- SxibPreScRC (4-byte field to contain an application defined return code).
- SxibPreScRS (4-byte field to contain an application defined reason code).

Before returning to the syscall caller, the z/OS UNIX copies these fields into the THLI control block. The syscall caller can then examine the THLI to determine why the syscall was rejected. These THLI fields are defined to have the same size and purpose as the corresponding Sxib fields that were previously specified.

- ThliRejectSCXProduct (12-byte field to identify the application or exit routine)
- ThliReject SCXReturnCode (4-byte field to contain an application defined return code)
- ThliRejectSCXReasonCode (4-byte field to contain an application defined reason code)

Post-syscall exit routines are run for syscalls that are rejected by a pre-syscall exit routine. The caller's return parameters (rv/rc/rs) are set to the values that were previously described and will be available for the post-syscall exit routines to examine.

Only syscalls whose last three parameters are return\_value, return\_code and reason\_code can be rejected by a pre-syscall exit routine. Syscalls that abend on failure (BPX1GPI – getpid()) cannot be rejected. The return code value from pre-syscall exit routines are ignored for those syscalls.

## Restrictions

1. Syscall exits are not supported for SRB mode callers. No syscall exit routines (pre or post) are invoked for SRB callers.
2. The BPX\_PRE\_SYSCALL and BPX\_POST\_SYSCALL exit points are limited to five exit routines each. The dynamic exits facility allows more than five exit routines to be added to the exit points but only the first five exit routines are invoked. The order that exit routines are invoked is controlled by the dynamic exits facility. For information about the order in which exit routines are invoked, see [CSVDYNEX - Provide dynamic exit services](#) in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.
3. The use of the following z/OS UNIX callable services are not allowed from syscall exit routines. z/OS UNIX has determined that the following callable services could negatively impact the system due to the caller's environment while syscall exit routines are running. If these callable services are called from a syscall exit routine, they will be unsuccessful. For syscalls that fail with a return value of -1 or 0, the return code will be set to EMVSERR and reason code to JrSyscallExit. Syscalls that abend on error will be abended with completion code EC6, reason code 'xxxxC056' x.

Syscall name	Stub name
aio_suspend()	BPX1ASP, BPX4ASP
alarm()	BPX1ALR, BPX4ALR
auth_check_rsrc_np()	BPX1ACK
chpriority()	BPX1CHP

<b>Syscall name</b>	<b>Stub name</b>
clone()	BPX1CLN, BPX4CLN
convert_id_np()	BPX1CID, BPX4CID
cond_cancel()	BPX1CCA, BPX4CCA
cond_setup()	BPX1CSE, BPX4CSE
cond_timed_wait()	BPX1CTW, BPX4CTW
cond_wait()	BPX1CWA, BPX4CWA
deletehfs()	BPX1DEL, BPX4DEL
exec()	BPX1EXC, BPX4EXC
exit()	BPX1EXI, BPX4EXI
execmvs()	BPX1EXM, BPX4EXM
extended_loadhfs()	BPX1LDX, BPX4LDX
fork()	BPX1FRK, BPX4FRK
getitimer()	BPX1GTR, BPX4GTR
istty()	BPX1ITY
loadhfs()	BPX1LOD, BPX4LOD
mmap()	BPX1MMP, BPX4MMP
mprotect()	BPX1MPR, BPX4MPR
munmap()	BPX1MUM
msync()	BPX1MSY
MVSIptAffinity	BPX1IPT, BPX4IPT
MVSpause()	BPX1MP, BPX4MP
mvsprocclp()	BPX1MPC, BPX4MPC
mvssigsetup()	BPX1MSS, BPX4MSS
MVSpauseinit()	BPX1MPI
MvsThreadAffinity()	BPX1TAF, BPX4TAF
mvsunsigsetup()	BPX1MSD, BPX4MSD
__map_init()	BPX1MMI, BPX4MMI
__map_service()	BPX1MMS, BPX4MMS
__osenv()	BPX1OSE, BPX4OSE
password()	BPX1PWD, BPX4PWD
pause()	BPX1PAS, BPX4PAS
pthread_quiesce()	BPX1PTQ, BPX4PTQ
pthread_quiesce _and_get_np()	BPX1PQG, BPX4PQG
pthread_security_np()	BPX1TLS, BPX4TLS
pthread_setintr	BPX1PSI, BPX4PSI)
pthread_setintrtrtype()	BPX1PST, BPX4PST
pthread_testintr()	BPX1PTI



<b>Syscall name</b>	<b>Stub name</b>
pthread_xandg()	BPX1PTX, BPX4PTX
ptrace()	BPX1PTR, BPX4PTR
__pid_affinity()	BPX1PAF, BPX4PAF
__poe()	BPX1POE, BPX4POE
quiesce()	BPX1QSE, BPX4QSE
nice()	BPX1NIC, BPX4NIC
send_file	BPX1SF, BPX4SF
send_filePC-only()	BPX1SEC, BPX4SEC
server_init()	BPX1SIN
server_pwu()	BPX1SPW, BPX4SPW
setdubdefault()	BPX1SDD, BPX4SDD
setegid()	BPX1SEG, BPX4SEG
seteuid()	BPX1SEU, BPX4SEU
setgid()	BPX1SGI, BPX4SGI
setgrent	BPX1SGE, BPX4SGE)
setgroups()	BPX1SGR, BPX4SGR
setitimer()	BPX1STR, BPX4STR
setpgid()	BPX1SPG, BPX4SPG
setpwent	BPX1SPE, BPX4SPE
setregid()	BPX1SRG, BPX4SRG
setreuid()	BPX1SRU, BPX4SRU
setsid()	BPX1SSI, BPX4SSI
set_timer_event()	BPX1STE, BPX4STE
setuid()	BPX1SUI, BPX4SUI
sigaction()	BPX1SIA, BPX4SIA
sigpending()	BPX1SIP, BPX4SIP
sigprocmask()	BPX1SPM, BPX4SPM
sigputback()	BPX1SPB, BPX4SPB
sigsuspend()	BPX1SSU, BPX4SSU
sigtimedwait()	BPX1STW, BPX4STW
sigwait()	BPX1SWT, BPX4SWT
sleep()	BPX1SLP, BPX4SLP
spawn()	BPX1SPN, BPX4SPN
srx_np	BPX1SRX, BPX4SRX
__security()	BPX1SIN, BPX4SIN
__shm_lock()	BPX1SLK, BPX4SLK
__sigactionset()	BPX1SA2, BPX4SA2
tcdrain()	BPX1TDR, BPX4TDR

Syscall name	Stub name
tcflow()	BPX1TFW, BPX4TFW
tcflush()	BPX1TFH, BPX4TFH
tcgetattr()	BPX1TGA, BPX4TGA
tcgetpgrp()	BPX1TGP, BPX4TGP
tcsendbreak()	BPX1TSB, BPX4TSB
tcsetattr()	BPX1TSA, BPX4TSA
tcsetpgrp()	BPX1TSP, BPX4TSP
ttynname()	BPX1TYN
unquiesce()	BPX1UQS, BPX4UQS
waitid()/wait3()	BPX1WTE
wait()	BPX1WAT, BPX4WAT
w_getpsent()	BPX1GPS
__wlm()	BPX1WLM, BPX4WLM

## Usage notes

1. All syscall exit routines are invoked with a z/OS UNIX non-space switched PC linkage stack. This PC is reserved for use by the z/OS UNIX kernel for syscall exits. Because the linkage stack is not a normal z/OS UNIX syscall linkage stack it cannot be modified by signals and it must be dequeued (PR) before returning to the caller. For this reason, signal delivery is disabled while syscall exit routines are running. Exit routines should not call any callable services that depend solely on a signal to be resumed. Care should be taken when using any allowed syscall that can receive a return code of EINTR because it could result in an unbounded wait.
2. Exit routines are responsible to free any resources they obtained before exiting.
3. All exit routines added to the pre-syscall and post-syscall exit points, up to the limit of the first five, are run each time the associated exit point is reached. This is especially relevant for the pre-syscall exit point because the Ibis field is shared between all exit routines. If multiple exit routines elect to reject a syscall, only the information from the last exit routine that rejected the syscall will be available in the Ibis field. Exits that choose not to reject a syscall should not alter the Sxib fields that identify the exit routine that rejected the syscall or the associated return code and reason code fields.
4. Post-syscall exit routines are not invoked for syscall that do not return to the caller or cause the caller to be undubbed. For example, exec and execMVS never return to the caller, mvsprocclp and exit can cause the caller to be undubbed.
5. IBM recommends that the z/OS UNIX file that is specified by the SC\_EXITTABLE statement reside in the /etc directory, which is consistent with the location of other configuration-related data. To prevent alteration by an unauthorized user, define the file permissions correctly.

---

## Appendix L. Callable service numbers

z/OS UNIX provides a C header, BPXYSYSC (syscalls.h). It defines constants that map the supported callable service numbers to their corresponding function names. These numbers represent an index into the z/OS UNIX callable service table and are used to calculate the offsets in [Appendix A, “System control offsets to callable services,”](#) on page 1045. This C header is located in SYS1.SIEAHDR.H as well as in the /usr/include/zos directory.

### BPXYSYSC (syscalls.h)

z/OS UNIX system call defines for C:

```
#define SYS_resource 1
#define SYS_isatty 2
#define SYS_ttynam 3
#define SYS_tcflush 4
#define SYS_tcdrain 5
#define SYS_tcflow 6
#define SYS_tcgetattr 7
#define SYS_tcgetpgrp 8
#define SYS_tcsetattr 9
#define SYS_tcsendbreak 10
#define SYS_tcsetpgrp 11
#define SYS_access 12
#define SYS_chdir 13
#define SYS_chmod 14
#define SYS_chown 15
#define SYS_closedir 16
#define SYS_close 17
#define SYS_w_getmntent 18
#define SYS_w_statvfs 19
#define SYS_chaudit 20
#define SYS_fchmod 21
#define SYS_fchown 22
#define SYS_fcntl 23
#define SYS_fpathconf 24
#define SYS_fstat 25
#define SYS_fsync 26
#define SYS_ftruncate 27
#define SYS_getcwd 28
#define SYS_w_ioctl 29
#define SYS_link 30
#define SYS_lseek 31
#define SYS_lstat 32
#define SYS_mkdir 33
#define SYS_fchmodit 34
#define SYS_mknod 35
#define SYS_mount 36
#define SYS_opendir 37
#define SYS_open 38
#define SYS_pathconf 39
#define SYS_pipe 40
#define SYS_readdir 41
#define SYS_readlink 42
#define SYS_read 43
#define SYS_rename 44
#define SYS_rewinddir 45
#define SYS_rmdir 46
#define SYS_stat 47
#define SYS_symlink 48
#define SYS_extlink_np 49
#define SYS_umask 50
#define SYS_umount 51
#define SYS_unlink 52
#define SYS_utime 53
#define SYS_write 54
#define SYS_alarm 55
#define SYS_exec 56
#define SYS_exit 57
#define SYS_execmvs 58
#define SYS_fork 59
#define SYS_getegid 60
```

#define SYS_geteuid	61
#define SYS_getgrgid	62
#define SYS_getgrnam	63
#define SYS_getgroups	64
#define SYS_getgid	65
#define SYS_getlogin	66
#define SYS_getpgrp	67
#define SYS_getpid	68
#define SYS_getpwnam	69
#define SYS_getppid	70
#define SYS_getpwuid	71
#define SYS_getugrps	72
#define SYS_getuid	73
#define SYS_setdubdefault	74
#define SYS_kill	76
#define SYS_mvssigsetup	77
#define SYS_pause	78
#define SYS_ptrace	79
#define SYS_sigaction	80
#define SYS_setgid	81
#define SYS_mvsunsigsetup	83
#define SYS_sigpending	84
#define SYS_sleep	85
#define SYS_setpgid	86
#define SYS_sigprocmask	87
#define SYS_setsid	88
#define SYS_sigsuspend	89
#define SYS_setuid	90
#define SYS_sysconf	91
#define SYS_times	92
#define SYS_uname	93
#define SYS_wait	94
#define SYS_quiesce	96
#define SYS_unquiesce	97
#define SYS_mvsiptaffinity	98
#define SYS_v_iocctl	99
#define SYS_mvsprocclp	101
#define SYS_pthread_quiesce	102
#define SYS_sigputback	103
#define SYS_seteuid	104
#define SYS_setegid	105
#define SYS_w_getpsent	106
#define SYS_pthread_create	107
#define SYS_pthread_exit_and_get	108
#define SYS_pthread_join	109
#define SYS_pthread_detach	110
#define SYS_pthread_cancel	111
#define SYS_pthread_self	112
#define SYS_pthread_setintr	114
#define SYS_pthread_kill	115
#define SYS_sigwait	116
#define SYS_pthread_setintrtype	117
#define SYS_pthread_testintr	118
#define SYS_cond_cancel	119
#define SYS_cond_post	120
#define SYS_cond_setup	121
#define SYS_cond_timed_wait	122
#define SYS_cond_wait	123
#define SYS_chattr	124
#define SYS_fchattr	125
#define SYS_accept	126
#define SYS_bind	127
#define SYS_connect	128
#define SYS_gethostid	129
#define SYS_getpeername	130
#define SYS_getsockopt	131
#define SYS_listen	132
#define SYS_readv	133
#define SYS_recv	134
#define SYS_recvfrom	135
#define SYS_recvmsg	136
#define SYS_select	137
#define SYS_send	138
#define SYS_sendmsg	139
#define SYS_sendto	140
#define SYS_setpeer	141
#define SYS_shutdown	142
#define SYS_socket_pair	143
#define SYS_writev	144
#define SYS_v_reg	145
#define SYS_v_rpn	146
#define SYS_bind2addrsel	147

#define SYS_v_get	148
#define SYS_v_rel	149
#define SYS_v_lookup	150
#define SYS_v_rdwrr	151
#define SYS_v_readaddr	152
#define SYS_v_readlink	153
#define SYS_v_create	154
#define SYS_v_mkdir	155
#define SYS_v_symlink	156
#define SYS_v_getattr	157
#define SYS_v_setattr	158
#define SYS_v_link	159
#define SYS_v_rmdir	160
#define SYS_v_remove	161
#define SYS_v_rename	162
#define SYS_v_fstatfs	163
#define SYS_v_lockctl	164
#define SYS_attach_exec	165
#define SYS_attach_execlmvs	166
#define SYS_mvspauseinit	169
#define SYS_set_thread_limits	170
#define SYS_mvspause	171
#define SYS_msgctl	172
#define SYS_msgget	173
#define SYS_msgrcv	174
#define SYS_msgsnd	175
#define SYS_semctl	176
#define SYS_semget	177
#define SYS_semop	178
#define SYS_shmat	179
#define SYS_shmctl	180
#define SYS_shmdt	181
#define SYS_shmget	182
#define SYS_w_getipc	183
#define SYS_setpriority	184
#define SYS_getpriority	185
#define SYS_nice	186
#define SYS_getitimer	187
#define SYS_setitimer	188
#define SYS_spawn	189
#define SYS_chpriority	190
#define SYS_pfsctl	191
#define SYS_getgrent	192
#define SYS_getpwent	193
#define SYS_setgrent	194
#define SYS_setpwent	195
#define SYS_password	196
#define SYS_setgroups	197
#define SYS_mmap	198
#define SYS_mprotect	199
#define SYS_msync	200
#define SYS_munmap	201
#define SYS_setrlimit	203
#define SYS_getrlimit	204
#define SYS_getrusage	205
#define SYS_truncate	206
#define SYS_lchown	207
#define SYS_wait_extension	209
#define SYS_statvfs	210
#define SYS_fstatvfs	211
#define SYS_fchdir	212
#define SYS_readaddr2	213
#define SYS_getpgid	214
#define SYS_getsid	215
#define SYS_sync	216
#define SYS_chroot	217
#define SYS_v_export	218
#define SYS_loadhfs	219
#define SYS_realpath	220
#define SYS_deletehfs	221
#define SYS_setreuid	222
#define SYS_setregid	223
#define SYS_tcgetcp	224
#define SYS_tcsetcp	225
#define SYS_tcsettables	226
#define SYS_tcgetsid	227
#define SYS_grantpt	228
#define SYS_unlockpt	229
#define SYS_ttyname2	230
#define SYS_isatty2	231
#define SYS_poll	232
#define SYS_getwd	233

#define SYS_read_extlink	234
#define SYS_v_access	235
#define SYS_querydub	236
#define SYS_oe_env_np	239
#define SYS_pthread_security_np	240
#define SYS_convert_id_np	241
#define SYS_auth_check_resource_np	242
#define SYS_recvmmsg2	243
#define SYS_sendmmsg2	244
#define SYS_w_pioctrl	245
#define SYS_asyncio	246
#define SYS_server_init	250
#define SYS_server_pwu	251
#define SYS_console_np	252
#define SYS_pthread_tag_np	253
#define SYS_getclientid	255
#define SYS_givesocket	256
#define SYS_takesocket	257
#define SYS___smf_record	258
#define SYS_v_pathconf	259
#define SYS___security	260
#define SYS___wlm	261
#define SYS_openstat	262
#define SYS___getthent	263
#define SYS_accept_and_recv	264
#define SYS_send_file	265
#define SYS___shm_lock	266
#define SYS___pid_affinity	267
#define SYS_set_timer_event	268
#define SYS_srx_np	269
#define SYS___sigactionset	270
#define SYS_aio_suspend	271
#define SYS_sigtimedwait	273
#define SYS___osenv	274
#define SYS_sigqueue	275
#define SYS_pread	276
#define SYS___smc	277
#define SYS_sw_sigdlv	280
#define SYS___mount	281
#define SYS___cpl	282
#define SYS___map_init	283
#define SYS___map_service	284
#define SYS_server_threads_query	285
#define SYS_mvstthreadaffinity	286
#define SYS_pthread_quiesce_and_get_np	287
#define SYS_gethostbyname	288
#define SYS_gethostbyaddr	289
#define SYS_getaddrinfo	290
#define SYS_freeaddrinfo	291
#define SYS_getnameinfo	292
#define SYS___poe	293
#define SYS_lchattr	294
#define SYS_v_open	295
#define SYS_v_close	296
#define SYS_loadhfs_ext	297
#define SYS_epoll_create	298
#define SYS_epoll_ctl	299
#define SYS_epoll_wait	300
#define SYS_inotify_init	301
#define SYS_inotify_add_watch	302
#define SYS_inotify_rm_watch	303
#define SYS_eventfd	304
#define SYS_prctl	305
#define SYS_accept4	306
#define SYS_pipe2	308
#define SYS_flock	309
#define SYS_openat	310
#define SYS_pivot_root	311
#define SYS_mkdirat	312
#define SYS_mknodat	313
#define SYS_readlinkat	314
#define SYS_symlinkat	315
#define SYS_fchownat	316
#define SYS_linkat	317
#define SYS_renameat2	318
#define SYS_faccessat	319
#define SYS_fchmodat	320
#define SYS_prlimit	321
#define SYS_clone	322
#define SYS_setns	323
#define SYS_unshare	324
#define SYS_sethostname	325

#define SYS_getrandom	326
#define SYS_syncfs	327
#define SYS_chattrat	328
#define SYS_fstatat	329
#define SYS_unlinkat	330
#define SYS___authenticate	331
#define SYS_MAXSYSCALL	331





---

## Appendix M. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.



## Notices

---

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Corporation  
Site Counsel  
2455 South Road*

Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### **COPYRIGHT LICENSE:**

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## **Terms and conditions for product documentation**

---

Permissions for the use of these publications are granted subject to the following terms and conditions.

### **Applicability**

These terms and conditions are in addition to any terms of use for the IBM website.

### **Personal use**

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### **Commercial use**

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

## **Rights**

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## **IBM Online Privacy Statement**

---

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at [ibm.com/privacy](http://ibm.com/privacy) and IBM's Online Privacy Statement at [ibm.com/privacy/details](http://ibm.com/privacy/details) in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at [ibm.com/software/info/product-privacy](http://ibm.com/software/info/product-privacy).

## **Policy for unsupported hardware**

---

Various z/OS elements, such as DFSMSdfp, JES2, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

## **Minimum supported hardware**

---

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

## Acknowledgments

---

InterOpen/POSIX Shell and Utilities is a source code product providing POSIX.2 (Shell and Utilities) functions to z/OS UNIX System Services. InterOpen/POSIX Shell and Utilities is developed and licensed by Mortice Kern Systems (MKS) Inc. of Waterloo, Ontario, Canada.

## Trademarks

---

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

# Index

## Special Characters

- \_\_authenticate (BPX1AUT))service  
example [1223](#)
- \_\_authenticate (BPX4AUT))service  
example [1317](#)
- \_\_console() (BPX1CCS, BPX4CCS) service [142](#)
- \_\_console() (BPX1CCS) service  
example [1225](#)
- \_\_console() (BPX4CCS) service  
example [1319](#)
- \_\_cpl (BPX1CPL) service [146](#)
- \_\_getthent (BPX1GTH, BPX4GTH) service [325](#)
- \_\_getthent (BPX1GTH) service  
example [1251](#)
- \_\_getthent (BPX4GTH) service  
example [1344](#)
- \_\_login (BPX1SEC) service  
example [1285](#)
- \_\_login (BPX4SEC) service  
example [1377](#)
- \_\_map\_init (BPX1MMI, BPX4MMI) service [412](#)
- \_\_map\_init (BPX1MMI) service  
example [1261](#)
- \_\_map\_init (BPX4MMI) service  
example [1354](#)
- \_\_map\_service (BPX1MMS, BPX4MMS) service [416](#)
- \_\_map\_service (BPX1MMS) service  
example [1262](#)
- \_\_map\_service (BPX4MMS) service  
example [1355](#)
- \_\_mount (BPX2MNT, BPX4MNT) service [447](#)
- \_\_mount (BPX2MNT) service  
example [1263](#)
- \_\_mount (BPX4MNT) service  
example [1356](#)
- \_\_passwd (BPX1PWD, BPX4PWD) service [530](#)
- \_\_passwd (BPX1PWD) service  
example [1276](#)
- \_\_passwd (BPX4PWD) service  
example [1368](#)
- \_\_pid\_affinity (BPX1PAF, BPX4PAF) service [548](#)
- \_\_pid\_affinity (BPX1PAF) service  
example [1268](#)
- \_\_pid\_affinity (BPX4PAF) service  
example [1360](#)
- \_\_poe (BPX1POE) service  
example [1270](#)
- \_\_poe (BPX4POE) service  
example [1362](#)
- \_\_poe() (BPX1POE, BPX4POE) service [559](#)
- \_\_sigactionset (BPX1SA2, BPX4SA2) service [839](#)
- \_\_sigactionset (BPX1SA2) service  
example [1284](#)
- \_\_sigactionset (BPX4SA2) service  
example [1376](#)
- \_\_wlm (BPX1WLM, BPX4WLM) service [1018](#)

- \_\_WLM (BPX1WLM) service  
example [1309](#)
- \_\_WLM (BPX4WLM) service  
example [1402](#)
- \_authenticate  
authenticate the specified user's credentials [69](#)
- \_authenticate (BPX1AUT, BPX4AUT)service [69](#)
- \_exit (BPX1EXI, BPX4EXI) service [180](#)
- \_exit (BPX1EXI) service  
example [1235](#)
- \_exit (BPX4EXI) service  
example [1328](#)

## Numerics

64-bit environment [8](#)

## A

- accept (BPX1ACP, BPX4ACP) service [13](#)
- accept (BPX1ACP) service  
example [1221](#)
- accept (BPX4ACP) service  
example [1315](#)
- accept\_and\_recv (BPX1ANR, BPX4ANR) service [19](#)
- accept\_and\_recv(BPX1ANR) service  
example [1222](#)
- accept\_and\_recv(BPX4ANR) service  
example [1316](#)
- accept4 service  
example [1220](#), [1221](#), [1314](#), [1315](#)
- Accept4\_Init (BPX1AC4, BPX4AC4)service [16](#)
- access
  - check file availability [24](#)
  - group database
    - by group ID [268](#)
    - by group name [271](#)
    - sequentially [266](#), [765](#)
  - to callable services [1](#)
  - user database
    - sequentially [302](#), [782](#)
    - user ID [308](#)
    - user name [305](#), [530](#)
- access (BPX1ACC, BPX4ACC) service [24](#)
- access (BPX1ACC) service  
example [1219](#)
- access (BPX4ACC) service  
example [1313](#)
- accessibility
  - contact IBM [1451](#)
- Addr\_Info structures
  - free [236](#)
- aio\_suspend (BPX1ASP, BPX4ASP) service [27](#)
- aio\_suspend (BPX1ASP) service  
example [1222](#)
- aio\_suspend (BPX4ASP) service  
example [1316](#)

- alarm (BPX1ALR, BPX4ALR) service [30](#)
- alarm (BPX1ALR) service
  - example [1221](#)
- alarm (BPX4ALR) service
  - example [1315](#)
- alarm, set [30](#)
- alias
  - get
    - of a host name [282](#)
    - of an IP address [279](#)
- appropriate privileges [8](#)
- assembler programming language
  - CALL syntax [1](#)
- assistive technologies [1451](#)
- asynchronous I/O request
  - wait for [27](#)
- asynchronous read
  - file [32](#)
- asynchronous serial data
  - break transmission [934](#)
- asynchronous write
  - file [32](#)
- asyncio (BPX1AIO, BPX4AIO) service [32](#)
- asyncio (BPX1AIO) service
  - example [1221](#)
- asyncio (BPX4AIO) service
  - example [1315](#)
- attach
  - to callable services [1](#)
- ATTACH macro
  - multiple task created for signals [1417](#)
- attach\_exec (BPX1ATX, BPX4ATX) service [48](#)
- attach\_exec (BPX1ATX) service
  - example [1223](#)
- attach\_exec (BPX4ATX) service
  - example [1317](#)
- attach\_execmvs (BPX1ATM, BPX4ATM) service [58](#)
- attach\_execmvs (BPX1ATM) service
  - example [1222](#)
- attach\_execmvs (BPX4ATM) service
  - example [1316](#)
- attaching to a process for debugging [633](#)
- attribute
  - obtain terminal [926](#)
  - set terminal [937](#)
- audit flags
  - change file [95](#)
  - change file by descriptor [199](#)
- auth\_check
  - resource
    - access [64](#)
- auth\_check\_resource\_np (BPX1ACK, BPX4ACK) service [64](#)
- auth\_check\_resource\_np (BPX1ACK) service
  - example [1220](#)
- auth\_check\_resource\_np (BPX4ACK) service
  - example [1314](#)
- authenticate
  - specified user's credentials [69](#)
- automatic conversion
  - control [216](#)
- availability
  - file system [444](#), [447](#), [975](#)

## B

- bind (BPX1BND, BPX4BND) service [76](#)
- bind (BPX1BND) service
  - example [1224](#)
- bind (BPX4BND) service
  - example [1318](#)
- bind with source address selection (BPX1BAS)
  - example [1224](#)
- bind with source address selection (BPX4BAS)
  - example [1318](#)
- bind2addrsl (BPX1BAS, BPX4BAS) service [79](#)
- blind dubbing [757](#)
- BPX1AC4 (accept4) service
  - example [1220](#), [1221](#)
- BPX1ACC (access) service
  - example [1219](#)
- BPX1ACC, BPX4ACC (access) service [24](#)
- BPX1ACK (auth\_check\_resource\_np) service
  - example [1220](#)
- BPX1ACK, BPX4ACK (auth\_check\_resource\_np) service [64](#)
- BPX1ACP (accept) service
  - example [1221](#)
- BPX1ACP, BPX4ACP (accept) service [13](#)
- BPX1AIO (asyncio) service
  - example [1221](#)
- BPX1AIO, BPX4AIO (asyncio) service [32](#)
- BPX1ALR (alarm) service
  - example [1221](#)
- BPX1ALR, BPX4ALR (alarm) service [30](#)
- BPX1ANR (accept\_and\_rcv) service
  - example [1222](#)
- BPX1ANR, BPX4ANR (accept\_and\_rcv) service [19](#)
- BPX1ASP (aio\_suspend) service
  - example [1222](#)
- BPX1ASP, BPX4ASP (aio\_suspend) service [27](#)
- BPX1ATM (attach\_execmvs) service
  - example [1222](#)
- BPX1ATM, BPX4ATM (attach\_execmvs) service [58](#)
- BPX1ATX (attach\_exec) service
  - example [1223](#)
- BPX1ATX, BPX4ATX (attach\_exec) service [48](#)
- BPX1BAS (bind with source address selection)
  - example [1224](#)
- BPX1BAS, BPX4BAS (bind2addrsl) service [79](#)
- BPX1BND (bind) service
  - example [1224](#)
- BPX1BND, BPX4BND (bind) service [76](#)
- BPX1CCA (cond\_cancel) service
  - example [1225](#)
- BPX1CCA, BPX4CCA (cond\_cancel) service [125](#)
- BPX1CCS (\_\_console()) service
  - example [1225](#)
- BPX1CCS, BPX4CCS (\_\_console()) service [142](#)
- BPX1CHA (chaudit) service
  - example [1225](#)
- BPX1CHA, BPX4CHA (chaudit) service [95](#)
- BPX1CHD (chdir) service
  - example [1226](#)
- BPX1CHD, BPX4CHD (chdir) service [98](#)
- BPX1CHM (chmod) service
  - example [1226](#)
- BPX1CHM, BPX4CHM (chmod) service [101](#)
- BPX1CHO (chown) service



BPX1CHO (chown) service *(continued)*  
     example [1226](#)  
 BPX1CHO, BPX4CHO (chown) service [104](#)  
 BPX1CHOA (fchownat) service  
     example [1226](#)  
 BPX1CHP (chpriority) service  
     example [1227](#)  
 BPX1CHP, BPX4CHP (chpriority) service [107](#)  
 BPX1CHR (chattr) service  
     example [1228](#)  
 BPX1CHR, BPX4CHR (chattr) service [82](#)  
 BPX1CLD (closedir) service  
     example [1229](#)  
 BPX1CLD, BPX4CLD (closedir) service [124](#)  
 BPX1CLN, BPX4CLN (clone service) [114](#)  
 BPX1CLO (close) service  
     example [1229](#)  
 BPX1CLO, BPX4CLO (close) service [121](#)  
 BPX1CMA (fchmodat) service  
     example [1227](#), [1240](#), [1321](#)  
 BPX1CMA, BPX4CMA (fchmodat) service [206](#)  
 BPX1COA, BPX4COA (fchownat) [213](#)  
 BPX1CON (connect) service  
     example [1229](#)  
 BPX1CON, BPX4CON (connect) service [139](#)  
 BPX1CPL (\_\_cpl) service [146](#)  
 BPX1CPO, BPX4CPO (cond\_post) service [127](#)  
 BPX1CPO(cond\_post) service  
     example [1230](#)  
 BPX1CRA (chatrat) service  
     example [1230](#)  
 BPX1CRA, BPX4CRA (chatrat) service [91](#)  
 BPX1CRT (chroot) service  
     example [1230](#), [1231](#)  
 BPX1CRT, BPX4CRT (chroot) service [111](#)  
 BPX1CSE (cond\_setup) service  
     example [1231](#)  
 BPX1CSE, BPX4CSE (cond\_setup) service [130](#)  
 BPX1CTW (cond\_timed\_wait) service  
     example [1231](#)  
 BPX1CTW, BPX4CTW (cond\_timed\_wait) service [132](#)  
 BPX1CWA (cond\_wait) service  
     example [1232](#)  
 BPX1CWA, BPX4CWA (cond\_wait) service [136](#)  
 BPX1DEL (deleteHFS) service  
     example [1232](#)  
 BPX1DEL, BPX4DEL (deletehfs) service [148](#)  
 BPX1DSD, BPX4DSD (sw\_sigdlv) service [899](#)  
 BPX1EFD (eventfd) service  
     example [1232](#)  
 BPX1EFD, BPX4BEPC (eventfd) service [160](#)  
 BPX1ENV (oe\_env\_np) service  
     example [1233](#)  
 BPX1ENV, BPX4ENV (oe\_env\_np) service [500](#)  
 BPX1EPC (epoll\_create) service  
     example [1233](#)  
 BPX1EPL (epoll\_epl) service  
     example [1233](#)  
 BPX1EPW (epoll\_wait) service  
     example [1234](#)  
 BPX1EXC (exec) service  
     example [1234](#)  
 BPX1EXC, BPX4EXC (exec) service [163](#)  
 BPX1EXI (\_exit) service

BPX1EXI (\_exit) service *(continued)*  
     example [1235](#)  
 BPX1EXI, BPX4EXI (\_exit) service [180](#)  
 BPX1EXM (execmvs) service  
     example [1235](#)  
 BPX1EXM, BPX4EXM (execmvs) service [175](#)  
 BPX1EXT (extlink\_np) service  
     example [1235](#)  
 BPX1EXT, BPX4EXT (extlink\_np) service [183](#)  
 BPX1FAA (faccessat) service  
     example [1235](#), [1329](#)  
 BPX1FAA, BPX4ACC (faccessat) service [186](#)  
 BPX1FAI (freeaddrinfo) service  
     example [1236](#)  
 BPX1FAI, BPX4FAI (freeaddrinfo) service [236](#)  
 BPX1FCA (fchmod) service  
     example [1236](#)  
 BPX1FCA, BPX4FCA (fchmod) service [199](#)  
 BPX1FCD (fchdir) service  
     example [1237](#)  
 BPX1FCD, BPX4FCD (fchdir) service [202](#)  
 BPX1FCM (fchmod) service  
     example [1237](#)  
 BPX1FCM, BPX4FCM (fchmod) service [204](#)  
 BPX1FCO (fchown) service  
     example [1237](#)  
 BPX1FCO, BPX4FCO (fchown) service [210](#)  
 BPX1FCR (fchattr) service  
     example [1237](#)  
 BPX1FCR, BPX4FCR (fchattr) service [190](#)  
 BPX1FCT (fcntl) service  
     example [1238](#)  
 BPX1FCT, BPX4FCT (fcntl) service [216](#)  
 BPX1FLK (flock) example) service  
     example [1239](#)  
 BPX1FPC (fpathconf) service  
     example [1239](#)  
 BPX1FPC, BPX4FPC (fpathconf) service [233](#)  
 BPX1FRK (fork) service  
     example [1239](#)  
 BPX1FRK, BPX4FRK (fork) service [227](#)  
 BPX1FSA (fstatat) service  
     example [1333](#)  
 BPX1FSA, BPX4FSA (fstatat) service [241](#)  
 BPX1FST (fststat) service  
     example [1239](#)  
 BPX1FST, BPX4FST (fststat) service [238](#)  
 BPX1FSY (fsync) service  
     example [1240](#)  
 BPX1FSY, BPX4FSY (fsync) service [247](#)  
 BPX1FTR (ftruncate) service  
     example [1240](#)  
 BPX1FTR, BPX4FTR (ftruncate) service [249](#)  
 BPX1FTV (fstatvfs) service  
     example [1240](#)  
 BPX1FTV, BPX4FTV (fstatvfs) service [244](#)  
 BPX1GAI (getaddrinfo) service  
     example [1241](#)  
 BPX1GAI, BPX4GAI (getaddrinfo) service [251](#)  
 BPX1GCL (getclientid) service  
     example [1241](#)  
 BPX1GCL, BPX4GCL (getclientid) service [258](#)  
 BPX1GCW (getcwd) service  
     example [1241](#)

BPX1GCW, BPX4GCW (getcwd) service [260](#)  
 BPX1GEG (getegid) service  
     example [1242](#)  
 BPX1GEG, BPX4GEG (getegid) service [263](#)  
 BPX1GEP (getpgid) service  
     example [1242](#)  
 BPX1GEP, BPX4GEP (getpgid) service [295](#)  
 BPX1GES (getsid) service  
     example [1242](#)  
 BPX1GES, BPX4GES (getsid) service [317](#)  
 BPX1GET (w\_getip) service  
     example [1242](#)  
 BPX1GET, BPX4GET (w\_getip) service [991](#)  
 BPX1GEU (geteuid) service  
     example [1243](#)  
 BPX1GEU, BPX4GEU (geteuid) service [264](#)  
 BPX1GGE (getgrent) service  
     example [1243](#)  
 BPX1GGE, BPX4GGE (getgrent) service [266](#)  
 BPX1GGI (getgrgid) service  
     example [1243](#)  
 BPX1GGI, BPX4GGI (getgrgid) service [268](#)  
 BPX1GGN (getgrnam) service  
     example [1244](#)  
 BPX1GGN, BPX4GGN (getgrnam) service [271](#)  
 BPX1GGR (getgroups) service  
     example [1244](#)  
 BPX1GGR, BPX4GGR (getgroups) service [274](#)  
 BPX1GHA (gethostbyaddr) service  
     example [1244](#)  
 BPX1GHA, BPX4GHA (gethostbyaddr) service [279](#)  
 BPX1GHN (gethostbyname) service  
     example [1245](#)  
 BPX1GHN, BPX4GHN (gethostbyname) service [282](#)  
 BPX1GID (getgid) service  
     example [1245](#)  
 BPX1GID, BPX4GID (getgid) service [265](#)  
 BPX1GIV (givesocket) service  
     example [1245](#)  
 BPX1GIV, BPX4GIV (givesocket) service [332](#)  
 BPX1GLG (getlogin) service  
     example [1246](#)  
 BPX1GLG, BPX4GLG (getlogin) service [289](#)  
 BPX1GMN (w\_getmntent) service  
     example [1246](#)  
 BPX1GMN, BPX4GMN (w\_getmntent) service [995](#)  
 BPX1GNI (getnameinfo) service  
     example [1246](#)  
 BPX1GNI, BPX4GNI (getnameinfo) service [291](#)  
 BPX1GNM (getpeername or getsockname) service  
     example [1247](#)  
 BPX1GNM, BPX4GNM (getsockname or getpeername)  
 service [319](#)  
 BPX1GPE (getpwent) service  
     example [1247](#)  
 BPX1GPE, BPX4GPE (getpwent) service [302](#)  
 BPX1GPG (getpgrp) service  
     example [1247](#)  
 BPX1GPG, BPX4GPG (getpgrp) service [297](#)  
 BPX1GPI (getpid) service  
     example [1248](#)  
 BPX1GPI, BPX4GPI (getpid) service [298](#)  
 BPX1GPN (getpwnam) service  
     example [1248](#)  
 BPX1GPN, BPX4GPN (getpwnam) service [305](#)  
 BPX1GPP (getppid) service  
     example [1248](#)  
 BPX1GPP, BPX4GPP (getppid) service [299](#)  
 BPX1GPS (w\_getpsent) service  
     example [1248](#)  
 BPX1GPT (grantpt) service  
     example [1249](#)  
 BPX1GPT, BPX4GPT (grantpt) service [336](#)  
 BPX1GPU (getpwuid) service  
     example [1249](#)  
 BPX1GPU, BPX4GPU (getpwuid) service [308](#)  
 BPX1GPY (getpriority) service  
     example [1249](#)  
 BPX1GPY, BPX4GPY (getpriority) service [300](#)  
 BPX1GRD (getrandom) service  
     example [1250](#)  
 BPX1GRL (getrlimit) service  
     example [1250](#)  
 BPX1GRL, BPX4GRL (getrlimit) service [313](#)  
 BPX1GRU (getrusage) service  
     example [1250](#)  
 BPX1GRU, BPX4GRU (getrusage) service [315](#)  
 BPX1GTH (\_\_getthent) service  
     example [1251](#)  
 BPX1GTH, BPX4GTH (\_\_getthent) service [325](#)  
 BPX1GTR (getitimer) service  
     example [1251](#)  
 BPX1GTR, BPX4GTR (getitimer) service [287](#)  
 BPX1GUG (getgroupsbyname) service  
     example [1251](#)  
 BPX1GUG, BPX4GUG (getgroupsbyname) service [276](#)  
 BPX1GUI (getuid) service  
     example [1252](#)  
 BPX1GUI, BPX4GUI (getuid) service [329](#)  
 BPX1GWD (getwd) service  
     example [1252](#)  
 BPX1GWD, BPX4GWD (getwd) service [330](#)  
 BPX1HST (gethostid or gethostname) service  
     example [1252](#)  
 BPX1HST, BPX4HST (gethostid or gethostname) service [285](#)  
 BPX1IAW (inotify\_add\_watch) service  
     example [1252](#)  
 BPX1IAW, BPX4IAW (inotify\_add\_watch) service [338](#)  
 BPX1INI (inotify\_init) service  
     example [1253](#), [1346](#)  
 BPX1INI, BPX4INI (inotify\_init) service [341](#)  
 BPX1IOC (w\_ioclt) service  
     example [1253](#)  
 BPX1IOC, BPX4IOC (w\_ioclt) service [1003](#)  
 BPX1IPT (mvsiptaffinity) service  
     example [1253](#)  
 BPX1IPT, BPX4IPT (mvsiptaffinity) service [476](#)  
 BPX1IRM (inotify\_rm\_watch) service  
     example [1254](#)  
 BPX1IRW, BPX4IRW (inotify\_rm\_watch) service [344](#)  
 BPX1ITY (isatty) service  
     example [1254](#)  
 BPX1KIL (kill) service  
     example [1254](#)  
 BPX1KIL, BPX4KIL (kill) service [360](#)  
 BPX1LCO (lchown) service  
     example [1255](#)  
 BPX1LCO, BPX4LCO (lchown) service [380](#)

BPX1LCR (lchattr) service  
     example [1255](#)  
 BPX1LCR, BPX4LCR (lchattr) service [371](#)  
 BPX1LDX (loadhfs extended) service  
     example [1255](#)  
 BPX1LDX, BPX4LDX (loadhfs extended) service [399](#)  
 BPX1LNA, BPX4LNA (linkat) service [387](#)  
 BPX1LNK (link) service  
     example [1258](#)  
 BPX1LNK, BPX4LNK (link) service [383](#)  
 BPX1LOD (loadhfs) service  
     example [1257](#)  
 BPX1LOD, BPX4LOD (loadhfs) service [393](#)  
 BPX1LSK (lseek) service  
     example [1258](#)  
 BPX1LSK, BPX4LSK (lseek) service [406](#)  
 BPX1LSN (listen) service  
     example [1258](#)  
 BPX1LSN, BPX4LSN (listen) service [391](#)  
 BPX1LST (lstat) service  
     example [1258](#)  
 BPX1LST, BPX4LST (lstat) service [409](#)  
 BPX1MAT (shmat) service  
     example [1259](#)  
 BPX1MAT, BPX4MAT (shmat) service [804](#)  
 BPX1MCT (shmctl) service  
     example [1259](#)  
 BPX1MCT, BPX4MCT (shmctl) service [809](#)  
 BPX1MDA (mkdirat) service  
     example [1259](#)  
 BPX1MDA, BPX4MKD (mkdirat) service [423](#)  
 BPX1MDT (shmdt) service  
     example [1260](#)  
 BPX1MDT, BPX4MDT (shmdt) service [812](#)  
 BPX1MGT (shmget) service  
     example [1260](#)  
 BPX1MGT, BPX4MGT (shmget) service [827](#)  
 BPX1MKD (mkdir) service  
     example [1260](#)  
 BPX1MKD, BPX4MKD (mkdir) service [420](#)  
 BPX1MKN (mknod) service  
     example [1261](#)  
 BPX1MKN, BPX4MKN (mknod) service [427](#)  
 BPX1MMI (\_\_map\_init) service  
     example [1261](#)  
 BPX1MMI, BPX4MMI (\_\_map\_init) service [412](#)  
 BPX1MMP (mmap) service  
     example [1262](#)  
 BPX1MMP, BPX4MMP (mmap) service [435](#)  
 BPX1MMS (\_\_map\_service) service  
     example [1262](#)  
 BPX1MMS, BPX4MMS (\_\_map\_service) service [416](#)  
 BPX1MNA (mknodat) service  
     example [1262](#)  
 BPX1MNA, BPX4MNA (mknodat) service [431](#)  
 BPX1MNT (mount) service  
     example [1263](#)  
 BPX1MP (mvspause) service  
     example [1263](#)  
 BPX1MP, BPX4MP (mvspause) service [479](#)  
 BPX1MPC (mvspocclp) service  
     example [1264](#)  
 BPX1MPC, BPX4MPC (mvspocclp) service [484](#)  
 BPX1MPI (mvspauseinit) service  
     example [1265](#)  
 BPX1MPI, BPX4MPI (mvspauseinit) service [481](#)  
 BPX1MPR (mprotect) service  
     example [1265](#)  
 BPX1MPR, BPX4MPR (mprotect) service [452](#)  
 BPX1MSD (mvsunsigsetup) service  
     example [1265](#)  
 BPX1MSD, BPX4MSD (mvsunsigsetup) service [495](#)  
 BPX1MSS (mvssigsetup) service  
     example [1266](#)  
 BPX1MSS, BPX4MSS (mvssigsetup) service [487](#)  
 BPX1MSY (msync) service  
     example [1266](#)  
 BPX1MSY, BPX4MSY (msync) service [469](#)  
 BPX1MUN (munmap) service  
     example [1266](#)  
 BPX1MUN, BPX4MUN (munmap) service [473](#)  
 BPX1NIC (nice) service  
     example [1266](#)  
 BPX1NIC, BPX4NIC (nice) service [497](#)  
 BPX1OAT, BPX4OAT (openat/openat4) service [518](#)  
 BPX1OPD (opendir) service  
     example [1267](#)  
 BPX1OPD, BPX4OPD (opendir) service [523](#)  
 BPX1OPN (open) service  
     example [1267](#)  
 BPX1OPN, BPX4OPN (open) service [511](#)  
 BPX1OPT (getsockopt or setsockopt) service  
     example [1268](#)  
 BPX1OPT, BPX4OPT (getsockopt or setsockopt) service [322](#)  
 BPX1PAF (\_\_pid\_affinity) service  
     example [1268](#)  
 BPX1PAF, BPX4PAF (\_\_pid\_affinity) service [548](#)  
 BPX1PAS (pause) service  
     example [1269](#)  
 BPX1PAS, BPX4PAS (pause) service [539](#)  
 BPX1PCF (pathconf) service  
     example [1269](#)  
 BPX1PCF, BPX4PCF (pathconf) service [535](#)  
 BPX1PCT (pfsctl) service  
     example [1269](#)  
 BPX1PCT, BPX4PCT (pfsctl) service [541](#)  
 BPX1PI2 (pipe2) service  
     example [1270](#)  
 BPX1PIO, BPX4PIO (w\_pioctl) service [1025](#)  
 BPX1PIP (pipe) service  
     example [1270](#)  
 BPX1PIP, BPX4PIP (pipe) service [551](#)  
 BPX1PLC (prctl) service  
     example [1271](#)  
 BPX1POE (\_\_poe) service  
     example [1270](#)  
 BPX1POE, BPX4POE (\_\_poe()) service [559](#)  
 BPX1POL (poll) service  
     example [1270](#)  
 BPX1POL, BPX4POL (poll) service [564](#)  
 BPX1PRC, BPX4PRC (prctl) service [568](#)  
 BPX1PRL (prlimit) service  
     example [1271](#), [1363](#), [1366](#)  
 BPX1PRL, BPX4PRL (prlimit) service [574](#)  
 BPX1PSI (pthread\_setintrl) service  
     example [1272](#)  
 BPX1PSI, BPX4PSI (pthread\_setintrl) service [610](#)

BPX1PST (pthread\_setinrttype) service  
example [1272](#)

BPX1PST, BPX4PST (pthread\_setinrttype) service [613](#)

BPX1PTB (pthread\_cancel) service  
example [1272](#)

BPX1PTB, BPX4PTB (pthread\_cancel) service [579](#)

BPX1PTC (pthread\_create) service  
example [1272](#)

BPX1PTC, BPX4PTC (pthread\_create) service [581](#)

BPX1PTD (pthread\_detach) service  
example [1273](#)

BPX1PTD, BPX4PTD (pthread\_detach) service [586](#)

BPX1PTI (pthread\_testintr) service  
example [1273](#)

BPX1PTI, BPX4PTI (pthread\_testintr) service [618](#)

BPX1PTJ (pthread\_join) service  
example [1273](#)

BPX1PTJ, BPX4PTJ (pthread\_join) service [592](#)

BPX1PTK (pthread\_kill) service  
example [1274](#)

BPX1PTK, BPX4PTK (pthread\_kill) service [595](#)

BPX1PTQ (pthread\_quiesce) service  
example [1274](#)

BPX1PTQ, BPX4PTQ (pthread\_quiesce) service [598](#)

BPX1PTR (ptrace) service  
example [1274](#)

BPX1PTR, BPX4PTR (ptrace) service [620](#)

BPX1PTS (pthread\_self) service  
example [1274](#)

BPX1PTS, BPX4PTS (pthread\_self) service [609](#)

BPX1PTT (pthread\_tag\_np) service  
example [1275](#)

BPX1PTT, BPX4PTT (pthread\_tag\_np) service [615](#)

BPX1PTX (pthread\_exit\_and\_get) service  
example [1275](#)

BPX1PTX, BPX4PTX (pthread\_exit\_and\_get) service [588](#)

BPX1PVR (pivot\_root) service  
example [1275](#)

BPX1PVR, BPX4PVR (change root mount) [556](#)

BPX1PWD (\_\_passwd) service  
example [1276](#)

BPX1PWD, BPX4PWD (\_\_passwd) service [530](#)

BPX1QCT (msgctl) service  
example [1276](#)

BPX1QCT, BPX4QCT (msgctl) service [455](#)

BPX1QDB (querydub) service  
example [1276](#)

BPX1QDB, BPX4QDB (querydub) service [646](#)

BPX1QGT (msgget) service  
example [1276](#)

BPX1QGT, BPX4QGT (msgget) service [458](#)

BPX1QRC (msgrcv) service  
example [1277](#)

BPX1QRC, BPX4QRC (msgrcv) service [462](#)

BPX1QSE (quiesce) service  
example [1277](#)

BPX1QSE, BPX4QSE (quiesce) service [650](#)

BPX1QSN (msgsnd) service  
example [1277](#)

BPX1QSN, BPX4QSN (msgsnd) service [465](#)

BPX1RCV (recv) service  
example [1278](#)

BPX1RCV, BPX4RCV (recv) service [680](#)

BPX1RD2 (readdir2) service

BPX1RD2 (readdir2) service (*continued*)  
example [1279](#)

BPX1RD2, BPX4RD2 (readdir2) service [659](#)

BPX1RDD (readdir) service  
example [1278](#)

BPX1RDD, BPX4RDD (readdir) service [656](#)

BPX1RDL (readlink) service  
example [1278](#)

BPX1RDL, BPX4RDL (readlink) service [666](#)

BPX1RDV (readv) service  
example [1279](#)

BPX1RDV, BPX4RDV (readv) service [673](#)

BPX1RDX (read\_extlink) service  
example [1279](#)

BPX1RDX, BPX4RDX (read\_extlink) service [663](#)

BPX1RED (read) service  
example [1280](#)

BPX1RED, BPX4RED (read) service [652](#)

BPX1REN (rename) service  
example [1280](#)

BPX1REN, BPX4REN (rename) service [690](#)

BPX1RFM (recvfrom) service  
example [1281](#)

BPX1RFM, BPX4RFM (recvfrom) service [683](#)

BPX1RLA (readlinkat) service  
example [1281](#)

BPX1RLA, BPX4RDL (readlink) service [669](#)

BPX1RMD (rmdir) service  
example [1281](#)

BPX1RMD, BPX4RMD (rmdir) service [703](#)

BPX1RMG (resource) service  
example [1281](#)

BPX1RMG, BPX4RMG (resource) service [699](#)

BPX1RNA, BPX4RNA (renameat) service [694](#)

BPX1RPH (realpath) service  
example [1283](#)

BPX1RPH, BPX4RPH (realpath) service [677](#)

BPX1RW (Pwrite) service  
example [1283](#)

BPX1RW, BPX4RW (pread() and pwrite()) service [572](#)

BPX1RWD (rewinddir) service  
example [1283](#)

BPX1RWD, BPX4RWD (rewinddir) service [701](#)

BPX1SA2 (\_\_sigactionset) service  
example [1284](#)

BPX1SA2, BPX4SA2 (\_\_sigactionset) service [839](#)

BPX1SCT (semctl) service  
example [1284](#)

BPX1SCT, BPX4SCT (semctl) service [713](#)

BPX1SDD (setdubdefault) service  
example [1284](#)

BPX1SDD, BPX4SDD (set\_dub\_default) service [752](#)

BPX1SEC (\_\_login) service  
example [1285](#)

BPX1SEC, BPX4SEC service [364](#)

BPX1SEG (setegid) service  
example [1285](#)

BPX1SEG, BPX4SEG (setegid) service [758](#)

BPX1SEL (select) service  
example [1285](#)

BPX1SEL, BPX4SEL (select) service [706](#)

BPX1SEU (seteuid) service  
example [1286](#)

BPX1SEU, BPX4SEU (seteuid) service [760](#)

BPX1SF (send\_file) service  
     example [1286](#)  
 BPX1SF, BPX4SF (send\_file) service [730](#)  
 BPX1SGE (setgrent) service  
     example [1286](#)  
 BPX1SGE, BPX4SGE (setgrent) service [765](#)  
 BPX1SGI (setgid) service  
     example [1287](#)  
 BPX1SGI, BPX4SGI (setgid) service [762](#)  
 BPX1SGQ (sigqueue) service  
     example [1287](#)  
 BPX1SGQ, BPX4SGQ (sigqueue) service [848](#)  
 BPX1SGR (setgroups) service  
     example [1287](#)  
 BPX1SGR, BPX4SGR (setgroups) service [766](#)  
 BPX1SGT (semget) service  
     example [1287](#)  
 BPX1SGT, BPX4SGT (semget) service [718](#)  
 BPX1SHT (shutdown) service  
     example [1288](#)  
 BPX1SHT, BPX4SHT (shutdown) service [832](#)  
 BPX1SIA (sigaction) service  
     example [1288](#)  
 BPX1SIA, BPX4SIA (sigaction) service [834](#)  
 BPX1SIN (server\_init) service  
     example [1288](#)  
 BPX1SIN, BPX4SIN (server\_init) service [742](#)  
 BPX1SIP (sigpending) service  
     example [1289](#)  
 BPX1SIP, BPX4SIP (sigpending) service [843](#)  
 BPX1SLK (shmem\_lock) service  
     example [1289](#)  
 BPX1SLK, BPX4SLK (shmem\_lock) service [814](#)  
 BPX1SLP (sleep) service  
     example [1289](#)  
 BPX1SLP, BPX4SLP (sleep) service [859](#)  
 BPX1SMC, BPX4SMC (shmem\_mutex\_condvar) service [819](#)  
 BPX1SMF (smf\_record) service  
     example [1290](#)  
 BPX1SMF, BPX4SMF (smf\_record) service [861](#)  
 BPX1SND (send) service  
     example [1291](#)  
 BPX1SND, BPX4SND (send) service [726](#)  
 BPX1SNS, BPX4SNS (unshare) service [771](#)  
 BPX1SOC (socket or socketpair) service  
     example [1291](#)  
 BPX1SOC, BPX4SOC (socket or socketpair) service [865](#)  
 BPX1SOP (semop) service  
     example [1292](#)  
 BPX1SOP, BPX4SOP (semop) service [723](#)  
 BPX1SPB (queue\_interrupt) service  
     example [1292](#)  
 BPX1SPB, BPX4SPB (queue\_interrupt) service [648](#)  
 BPX1SPE (setpwent) service  
     example [1292](#)  
 BPX1SPE, BPX4SPE (setpwent) service [782](#)  
 BPX1SPG (setpgid) service  
     example [1293](#)  
 BPX1SPG, BPX4SPG (setpgid) service [777](#)  
 BPX1SPM (sigprocmask) service  
     example [1293](#)  
 BPX1SPM, BPX4SPM (sigprocmask) service [845](#)  
 BPX1SPN (spawn) service  
     example [1293](#)  
 BPX1SPN, BPX4SPN (spawn) service [869](#)  
 BPX1SPR (setpeer) service  
     example [1294](#)  
 BPX1SPR, BPX4SPR (setpeer) service [775](#)  
 BPX1SPW (server\_pwu) service  
     example [1294](#)  
 BPX1SPW, BPX4SPW (server\_pwu) service [746](#)  
 BPX1SPY (setpriority) service  
     example [1295](#)  
 BPX1SPY, BPX4SPY (setpriority) service [779](#)  
 BPX1SRG (setregid) service  
     example [1295](#)  
 BPX1SRG, BPX4SRG (setregid) service [784](#)  
 BPX1SRL (setrlimit) service  
     example [1295](#)  
 BPX1SRL, BPX4SRL (setrlimit) service [789](#)  
 BPX1SRU (setreuid) service  
     example [1296](#)  
 BPX1SRU, BPX4SRU (setreuid) service [786](#)  
 BPX1SRX (srx\_np) service  
     example [1296](#)  
 BPX1SRX, BPX4SRX (srx\_np) service [886](#)  
 BPX1SSI (setsid) service  
     example [1296](#)  
 BPX1SSI, BPX4SSI (setsid) service [793](#)  
 BPX1SSU (sigsuspend) service  
     example [1297](#)  
 BPX1SSU, BPX4SSU (sigsuspend) service [851](#)  
 BPX1STA (stat) service  
     example [1297](#)  
 BPX1STA, BPX4STA (stat) service [893](#)  
 BPX1STE (set\_timer\_event) service  
     example [1297](#)  
 BPX1STE, BPX4STE (set\_timer\_event) service [798](#)  
 BPX1STF (w\_statvfs) service  
     example [1297](#)  
 BPX1STF, BPX4STF (w\_statvfs) service [1029](#)  
 BPX1STL (set\_thread\_limits) service  
     example [1298](#)  
 BPX1STL, BPX4STL (set\_thread\_limits) service [795](#)  
 BPX1STO (sendto) service  
     example [1298](#)  
 BPX1STO, BPX4STO (sendto) service [738](#)  
 BPX1STR (setitimer) service  
     example [1298](#)  
 BPX1STR, BPX4STR (setitimer) service [768](#)  
 BPX1STV (statvfs) service  
     example [1299](#)  
 BPX1STV, BPX4STV (statvfs) service [896](#)  
 BPX1STW (sigtimedwait) service  
     example [1299](#)  
 BPX1STW, BPX4STW (sigtimedwait) service [854](#)  
 BPX1SUI (setuid) service  
     example [1299](#)  
 BPX1SUI, BPX4SUI (setuid) service [800](#)  
 BPX1SWT (sigwait) service  
     example [1300](#)  
 BPX1SWT, BPX4SWT (sigwait) service [857](#)  
 BPX1SYA (symlinkat) service  
     example [1300](#)  
 BPX1SYA, BPX4SYA (symlinkat) service [905](#)  
 BPX1SYC (sysconf) service  
     example [1300](#)  
 BPX1SYC, BPX4SYC (sysconf) service [913](#)



BPX1SYF (syncfs) service  
     example [1300](#)  
 BPX1SYF, BPX4SYF (syncfs) service [912](#)  
 BPX1SYM (symlink) service  
     example [1301](#)  
 BPX1SYM, BPX4SYM (symlink) service [900](#)  
 BPX1SYN (sync) service  
     example [1301](#)  
 BPX1SYN, BPX4SYN (sync) service [910](#)  
 BPX1TAF (MVSThreadAffinity) service  
     example [1301](#)  
 BPX1TAF, BPX4TAF (MVSThreadAffinity) service [492](#)  
 BPX1TAK (takesocket) service  
     example [1301](#)  
 BPX1TAK, BPX4TAK (takesocket) service [916](#)  
 BPX1TDR (tcdrain) service  
     example [1302](#)  
 BPX1TDR, BPX4TDR (tcdrain) service [918](#)  
 BPX1TFH (tcflush) service  
     example [1302](#)  
 BPX1TFH, BPX4TFH (tcflush) service [923](#)  
 BPX1TFW (tcflow) service  
     example [1302](#)  
 BPX1TFW, BPX4TFW (tcflow) service [921](#)  
 BPX1TGA (tcgetattr) service  
     example [1302](#)  
 BPX1TGA, BPX4TGA (tcgetattr) service [926](#)  
 BPX1TGC (tcgetcp) service  
     example [1303](#)  
 BPX1TGC, BPX4TGC (tcgetcp) service [928](#)  
 BPX1TGP (tcgetpgrp) service  
     example [1303](#)  
 BPX1TGP, BPX4TGP (tcgetpgrp) service [931](#)  
 BPX1TGS (tcgetsid) service  
     example [1303](#)  
 BPX1TGS, BPX4TGS (tcgetsid) service [933](#)  
 BPX1TIM (times) service  
     example [1303](#)  
 BPX1TIM, BPX4TIM (times) service [950](#)  
 BPX1TLS (pthread\_security\_np) service  
     example [1304](#)  
 BPX1TLS, BPX4TLS (pthread\_security\_np) service [601](#)  
 BPX1TRU (truncate) service  
     example [1304](#)  
 BPX1TRU, BPX4TRU (truncate) service [953](#)  
 BPX1TSA (tcsetattr) service  
     example [1304](#)  
 BPX1TSA, BPX4TSA (tcsetattr) service [937](#)  
 BPX1TSB (tcsendbreak) service  
     example [1305](#)  
 BPX1TSB, BPX4TSB (tcsendbreak) service [934](#)  
 BPX1TSC (tcsetcp) service  
     example [1305](#)  
 BPX1TSC, BPX4TSC (tcsetcp) service [940](#)  
 BPX1TSP (tcsetpgrp) service  
     example [1305](#)  
 BPX1TSP, BPX4TSP (tcsetpgrp) service [944](#)  
 BPX1TST (tcsettables) service  
     example [1305](#)  
 BPX1TST, BPX4TST (tcsettables) service [946](#)  
 BPX1TYN (ttyname) service  
     example [1306](#)  
 BPX1TYN, BPX4TYN (ttyname) service [956](#)  
 BPX1ULA, BPX4ULA (unlinkat) service [969](#)  
 BPX1UMK (umask) service  
     example [1307](#)  
 BPX1UMK, BPX4UMK (umask) service [960](#)  
 BPX1UMT (umount) service  
     example [1307](#)  
 BPX1UMT, BPX4UMT (umount) service [961](#)  
 BPX1UNA (uname) service  
     example [1307](#)  
 BPX1UNA, BPX4UNA (uname) service [964](#)  
 BPX1UNL (unlink) service  
     example [1308](#)  
 BPX1UNL, BPX4UNL (unlink) service [966](#)  
 BPX1UNS (unshare) service  
     example [1308](#)  
 BPX1UNS, BPX4UNS (unshare service [977](#)  
 BPX1UPT (unlockpt) service  
     example [1308](#)  
 BPX1UPT, BPX4UPT (unlockpt) service [973](#)  
 BPX1UQS (unquiesce) service  
     example [1308](#)  
 BPX1UQS, BPX4UQS (unquiesce) service [975](#)  
 BPX1UTI (utime) service  
     example [1309](#)  
 BPX1UTI, BPX4UTI (utime) service [981](#)  
 BPX1WAT (wait) service  
     example [1309](#)  
 BPX1WAT, BPX4WAT (wait) service [984](#)  
 BPX1WLM (\_\_WLM) service  
     example [1309](#)  
 BPX1WLM, BPX4WLM (\_\_wlm) service [1018](#)  
 BPX1WRT (write) service  
     example [1310](#)  
 BPX1WRT, BPX4WRT (write) service [1032](#)  
 BPX1WRV (writev) service  
     example [1310](#)  
 BPX1WRV, BPX4WRV (writev) service [1036](#)  
 BPX1WTE (wait extension) service  
     example [1310](#)  
 BPX1WTE, BPX4WTE (wait-extension) service [987](#)  
 BPX1xxx module [2](#)  
 BPX2ITY (isatty) service  
     example [1254](#)  
 BPX2ITY, BPX4ITY (isatty) service [359](#)  
 BPX2MNT (\_\_mount) service  
     example [1263](#)  
 BPX2MNT, BPX4MNT (\_\_mount) service [447](#)  
 BPX2OPN, BPX4OPS (openstat) service [526](#)  
 BPX2OPT (open) service  
     example [1267](#)  
 BPX2RMS (recvmsg) service  
     example [1282](#)  
 BPX2RMS, BPX4RMS (recvmsg) service [687](#)  
 BPX2SMS (sendmsg) service  
     example [1290](#)  
 BPX2SMS, BPX4SMS (sendmsg) service [734](#)  
 BPX2TYN (ttyname) service  
     example [1306](#)  
 BPX2TYN, BPX4TYN (ttyname) service [958](#)  
 BPX2xxx module [2](#)  
 BPX4AC4 (accept4) service  
     example [1314](#), [1315](#)  
 BPX4ACC (access) service  
     example [1313](#)  
 BPX4ACK (auth\_check\_resource\_np) service

BPX4ACK (auth\_check\_resource\_np) service (*continued*)  
 example [1314](#)

BPX4ACP (accept) service  
 example [1315](#)

BPX4AIO (asyncio) service  
 example [1315](#)

BPX4ALR (alarm) service  
 example [1315](#)

BPX4ANR (accept\_and\_rcv) service  
 example [1316](#)

BPX4ASP (aio\_suspend) service  
 example [1316](#)

BPX4ATM (attach\_execmvs) service  
 example [1316](#)

BPX4ATX (attach\_exec) service  
 example [1317](#)

BPX4BAS (bind with source address selection)  
 example [1318](#)

BPX4BND (bind) service  
 example [1318](#)

BPX4CCA (cond\_cancel) service  
 example [1318](#)

BPX4CCS (\_\_console()) service  
 example [1319](#)

BPX4CHA (chaudit) service  
 example [1319](#)

BPX4CHD (chdir) service  
 example [1319](#)

BPX4CHM (chmod) service  
 example [1319](#)

BPX4CHO (chown) service  
 example [1320](#)

BPX4CHOA (fchownat) service  
 example [1320](#)

BPX4CHP (chpriority) service  
 example [1320](#)

BPX4CHR (chattr) service  
 example [1321](#)

BPX4CLD (closedir) service  
 example [1322](#)

BPX4CLO (close) service  
 example [1323](#)

BPX4CON (connect) service  
 example [1323](#)

BPX4CPO(cond\_post) service  
 example [1323](#)

BPX4CRA (chattrat) service  
 example [1323](#)

BPX4CRT (chroot) service  
 example [1324](#)

BPX4CSE (cond\_setup) service  
 example [1324](#)

BPX4CTW (cond\_timed\_wait) service  
 example [1325](#)

BPX4CWA (cond\_wait) service  
 example [1325](#)

BPX4DEL (deleteHFS) service  
 example [1325](#)

BPX4EFD (eventfd) service  
 example [1326](#)

BPX4ENV (oe\_env\_np) service  
 example [1326](#)

BPX4EPC (epoll\_create) service  
 example [1326](#)

BPX4EPL (epoll\_ctl) service  
 example [1327](#)

BPX4EPW (epoll\_wait) service  
 example [1327](#)

BPX4EXC (exec) service  
 example [1327](#)

BPX4EXI (\_exit) service  
 example [1328](#)

BPX4EXM (execmvs) service  
 example [1328](#)

BPX4EXT (extlink\_np) service  
 example [1329](#)

BPX4FAI (freeaddrinfo) service  
 example [1329](#)

BPX4FCA (fchaudit) service  
 example [1330](#)

BPX4FCD (fchdir) service  
 example [1330](#)

BPX4FCM (fchmod) service  
 example [1330](#)

BPX4FCO (fchown) service  
 example [1330](#)

BPX4FCR (fchattr) service  
 example [1331](#)

BPX4FCT (fcntl) service  
 example [1331](#)

BPX4FLK (flock) example) service  
 example [1332](#)

BPX4FPC (fpathconf) service  
 example [1332](#)

BPX4FRK (fork) service  
 example [1332](#)

BPX4FST (fstat) service  
 example [1333](#)

BPX4FSY (fsync) service  
 example [1333](#)

BPX4FTR (ftruncate) service  
 example [1334](#)

BPX4FTV (fstatvfs) service  
 example [1334](#)

BPX4GAI (getaddrinfo) service  
 example [1334](#)

BPX4GCL (getclientid) service  
 example [1335](#)

BPX4GCW (getcwd) service  
 example [1335](#)

BPX4GEG (getegid) service  
 example [1335](#)

BPX4GEP (getpgid) service  
 example [1335](#)

BPX4GES (getsid) service  
 example [1336](#)

BPX4GET (w\_getipc) service  
 example [1336](#)

BPX4GEU (geteuid) service  
 example [1336](#)

BPX4GGE (getgrent) service  
 example [1336](#)

BPX4GGI (getgrgid) service  
 example [1337](#)

BPX4GGN (getgrnam) service  
 example [1337](#)

BPX4GGR (getgroups) service  
 example [1338](#)

BPX4GHA (gethostbyaddr) service example <a href="#">1338</a>	BPX4LCR (lchatrr) service example <a href="#">1348</a>
BPX4GHN (gethostbyname) service example <a href="#">1338</a>	BPX4LDX (loadhfs extended) service example <a href="#">1348</a>
BPX4GID (getgid) service example <a href="#">1339</a>	BPX4LNK (link) service example <a href="#">1351</a>
BPX4GIV (givesocket) service example <a href="#">1339</a>	BPX4LOD (loadhfs) service example <a href="#">1350</a>
BPX4GLG (getlogin) service example <a href="#">1339</a>	BPX4LSK (lseek) service example <a href="#">1351</a>
BPX4GMN (w_getmntent) service example <a href="#">1340</a>	BPX4LSN (listen) service example <a href="#">1351</a>
BPX4GNI (getnameinfo) service example <a href="#">1340</a>	BPX4LST (lstat) service example <a href="#">1351</a>
BPX4GNM (getpeername or getsockname) service example <a href="#">1341</a>	BPX4MAT (shmat) service example <a href="#">1352</a>
BPX4GPE (getpwent) service example <a href="#">1341</a>	BPX4MCT (shmctl) service example <a href="#">1352</a>
BPX4GPG (getpgrp) service example <a href="#">1340</a>	BPX4MDA (mkdirat) service example <a href="#">1352</a>
BPX4GPI (getpid) service example <a href="#">1341</a>	BPX4MDT (shmdt) service example <a href="#">1353</a>
BPX4GPN (getpwnam) service example <a href="#">1341</a>	BPX4MGT (shmget) service example <a href="#">1353</a>
BPX4GPP (getppid) service example <a href="#">1342</a>	BPX4MKD (mkdir) service example <a href="#">1353</a>
BPX4GPT (grantpt) service example <a href="#">1342</a>	BPX4MKN (mknod) service example <a href="#">1354</a>
BPX4GPU (getpwuid) service example <a href="#">1342</a>	BPX4MMI (__map_init) service example <a href="#">1354</a>
BPX4GPY (getpriority) service example <a href="#">1342</a>	BPX4MMP (mmap) service example <a href="#">1355</a>
BPX4GRD (getrandom))service example <a href="#">1343</a>	BPX4MMS (__map_service) service example <a href="#">1355</a>
BPX4GRL (getrlimit) service example <a href="#">1343</a>	BPX4MNA (mknodat) service example <a href="#">1355</a>
BPX4GRU (getrusage) service example <a href="#">1343</a>	BPX4MNT (__mount) service example <a href="#">1356</a>
BPX4GTH (__getthent) service example <a href="#">1344</a>	BPX4MP (mvspause) service example <a href="#">1356</a>
BPX4GTR (getitimer) service example <a href="#">1344</a>	BPX4MPC (mvsprocclp) service example <a href="#">1356</a>
BPX4GUG (getgroupsbyname) service example <a href="#">1344</a>	BPX4MPI (mvspauseinit) service example <a href="#">1357</a>
BPX4GUI (getuid) service example <a href="#">1345</a>	BPX4MPR (mprotect) service example <a href="#">1357</a>
BPX4GWD (getwd) service example <a href="#">1345</a>	BPX4MSD (mvsunsigsetup) service example <a href="#">1357</a>
BPX4HST (gethostid or gethostname) service example <a href="#">1345</a>	BPX4MSS (mvssigsetup) service example <a href="#">1358</a>
BPX4IAW (inotify_add_watch) service example <a href="#">1346</a>	BPX4MSY (msync) service example <a href="#">1358</a>
BPX4IOC (w_ioctl) service example <a href="#">1347</a>	BPX4MUN (munmap) service example <a href="#">1358</a>
BPX4IPT (mvsiptaffinity) service example <a href="#">1347</a>	BPX4NIC (nice) service example <a href="#">1358</a>
BPX4IRM (inotify_rm_watch) service example <a href="#">1346</a>	BPX4OPD (opendir) service example <a href="#">1359</a>
BPX4ITY (isatty) service example <a href="#">1347</a>	BPX4OPN (open) service example <a href="#">1359</a>
BPX4KIL (kill) service example <a href="#">1347</a>	BPX4OPS (open) service example <a href="#">1359</a>
BPX4LCO (lchown) service example <a href="#">1348</a>	BPX4OPT (getsockopt or setsockopt) service example <a href="#">1360</a>



BPX4PAF (__pid_affinity) service example <a href="#">1360</a>	BPX4RDD (readdir) service example <a href="#">1371</a>
BPX4PAS (pause) service example <a href="#">1361</a>	BPX4RDL (readlink) service example <a href="#">1371</a>
BPX4PCF (pathconf) service example <a href="#">1361</a>	BPX4RDV (readv) service example <a href="#">1371</a>
BPX4PCT (pfctl) service example <a href="#">1361</a>	BPX4RDX (read extlink) service example <a href="#">1372</a>
BPX4PI2 (pipe2) service example <a href="#">1362</a>	BPX4RED (read) service example <a href="#">1372</a>
BPX4PIP (pipe) service example <a href="#">1362</a>	BPX4REN (rename) service example <a href="#">1373</a>
BPX4PLC (prctl) service example <a href="#">1363</a>	BPX4RFM (recvfrom) service example <a href="#">1373</a>
BPX4POE (__poe) service example <a href="#">1362</a>	BPX4RLA (readlinkat) service example <a href="#">1373</a>
BPX4POL (poll) service example <a href="#">1362</a>	BPX4RMD (rmdir) service example <a href="#">1374</a>
BPX4PSI (pthread_setinr) service example <a href="#">1364</a>	BPX4RMG (resource) service example <a href="#">1374</a>
BPX4PST (pthread_setinrtype) service example <a href="#">1364</a>	BPX4RMS (recvmsg) service example <a href="#">1374</a>
BPX4PTB (pthread_cancel) service example <a href="#">1364</a>	BPX4RPH (realpath) service example <a href="#">1375</a>
BPX4PTC (pthread_create) service example <a href="#">1364</a>	BPX4RW (Pwrite) service example <a href="#">1375</a>
BPX4PTD (pthread_detach) service example <a href="#">1365</a>	BPX4RWD (rewinddir) service example <a href="#">1376</a>
BPX4PTI (pthread_testintr) service example <a href="#">1365</a>	BPX4SA2 (__sigactionset) service example <a href="#">1376</a>
BPX4PTJ (pthread_join) service example <a href="#">1365</a>	BPX4SCT (semctl) service example <a href="#">1376</a>
BPX4PTK (pthread_kill) service example <a href="#">1366</a>	BPX4SDD (setdubdefault) service example <a href="#">1377</a>
BPX4PTQ (pthread_quiesce) service example <a href="#">1366</a>	BPX4SEC (__login) service example <a href="#">1377</a>
BPX4PTR (ptrace) service example <a href="#">1366</a>	BPX4SEG (setegid) service example <a href="#">1378</a>
BPX4PTS (pthread_self) service example <a href="#">1367</a>	BPX4SEL (select) service example <a href="#">1378</a>
BPX4PTT (pthread_tag_np) service example <a href="#">1367</a>	BPX4SEU (seteuid) service example <a href="#">1378</a>
BPX4PTX (pthread_exit_and_get) service example <a href="#">1367</a>	BPX4SF (send_file) service example <a href="#">1379</a>
BPX4PVR (pivot_root) service example <a href="#">1368</a>	BPX4SGE (setgrent) service example <a href="#">1379</a>
BPX4PWD (__passwd) service example <a href="#">1368</a>	BPX4SGI (setgid) service example <a href="#">1379</a>
BPX4QCT (msgctl) service example <a href="#">1368</a>	BPX4SGQ (sigqueue) service example <a href="#">1379</a>
BPX4QDB (querydub) service example <a href="#">1369</a>	BPX4SGR (setgroups) service example <a href="#">1380</a>
BPX4QGT (msgget) service example <a href="#">1369</a>	BPX4SGT (semget) service example <a href="#">1380</a>
BPX4QRC (msgrcv) service example <a href="#">1369</a>	BPX4SHT (shutdown) service example <a href="#">1380</a>
BPX4QSE (quiesce) service example <a href="#">1370</a>	BPX4SIA (sigaction) service example <a href="#">1381</a>
BPX4QSN (msgsnd) service example <a href="#">1370</a>	BPX4SIN (server_init) service example <a href="#">1381</a>
BPX4RCV (recv) service example <a href="#">1370</a>	BPX4SIP (sigpending) service example <a href="#">1381</a>
BPX4RD2 (readdir2) service example <a href="#">1372</a>	BPX4SLK (shmlock) service example <a href="#">1382</a>

BPX4SLP (sleep) service example <a href="#">1382</a>	BPX4SYF (syncfs) service example <a href="#">1393</a>
BPX4SMF (smf_record) service example <a href="#">1382</a>	BPX4SYM (symlink) service example <a href="#">1393</a>
BPX4SMS (sendmsg) service example <a href="#">1383</a>	BPX4SYN (sync) service example <a href="#">1394</a>
BPX4SND (send) service example <a href="#">1383</a>	BPX4TAF (MVSThreadAffinity) service example <a href="#">1394</a>
BPX4SOC (socket or socketpair) service example <a href="#">1384</a>	BPX4TAK (takesocket) service example <a href="#">1394</a>
BPX4SOP (semop) service example <a href="#">1384</a>	BPX4TDR (tcdrain) service example <a href="#">1394</a>
BPX4SPB (queue_interrupt) service example <a href="#">1385</a>	BPX4TFH (tcflush) service example <a href="#">1395</a>
BPX4SPE (setpwtent) service example <a href="#">1385</a>	BPX4TFW (tcflow) service example <a href="#">1395</a>
BPX4SPG (setpgid) service example <a href="#">1385</a>	BPX4TGA (tcgetattr) service example <a href="#">1395</a>
BPX4SPM (sigprocmask) service example <a href="#">1385</a>	BPX4TGC (tcgetcp) service example <a href="#">1395</a>
BPX4SPN (spawn) service example <a href="#">1386</a>	BPX4TGP (tcgetpgrp) service example <a href="#">1396</a>
BPX4SPR (setpeer) service example <a href="#">1387</a>	BPX4TGS (tcgetsid) service example <a href="#">1396</a>
BPX4SPW (server_pwu) service example <a href="#">1387</a>	BPX4TIM (times) service example <a href="#">1396</a>
BPX4SPY (setpriority) service example <a href="#">1387</a>	BPX4TLS (pthread_security_np) service example <a href="#">1396</a>
BPX4SRG (setregid) service example <a href="#">1388</a>	BPX4TRU (truncate) service example <a href="#">1397</a>
BPX4SRL (setrlimit) service example <a href="#">1388</a>	BPX4TSA (tcsetattr) service example <a href="#">1397</a>
BPX4SRU (setreuid) service example <a href="#">1388</a>	BPX4TSB (tcsendbreak) service example <a href="#">1397</a>
BPX4SRX (srx_np) service example <a href="#">1389</a>	BPX4TSC (tcsetcp) service example <a href="#">1397</a>
BPX4SSI (setsid) service example <a href="#">1389</a>	BPX4TSP (tcsetpgrp) service example <a href="#">1398</a>
BPX4SSU (sigsuspend) service example <a href="#">1389</a>	BPX4TST (tcsettables) service example <a href="#">1398</a>
BPX4STA (stat) service example <a href="#">1389</a>	BPX4TYN (ttyname) service example <a href="#">1399</a>
BPX4STE (set_timer_event) service example <a href="#">1390</a>	BPX4UMK (umask) service example <a href="#">1399</a>
BPX4STF (w_statvfs) service example <a href="#">1390</a>	BPX4UMT (umount) service example <a href="#">1399</a>
BPX4STL (set_thread_limits) service example <a href="#">1390</a>	BPX4UNA (uname) service example <a href="#">1400</a>
BPX4STO (sendto) service example <a href="#">1391</a>	BPX4UNL (unlink) service example <a href="#">1400</a>
BPX4STR (setitimer) service example <a href="#">1391</a>	BPX4UNS (unshare) service example <a href="#">1401</a>
BPX4STV (statvfs) service example <a href="#">1391</a>	BPX4UPT (unlockpt) service example <a href="#">1400</a>
BPX4STW (sigtimedwait) service example <a href="#">1392</a>	BPX4UQS (unquiesce) service example <a href="#">1400</a>
BPX4SUI (setuid) service example <a href="#">1392</a>	BPX4UTI (utime) service example <a href="#">1401</a>
BPX4SWT (sigwait) service example <a href="#">1392</a>	BPX4WAT (wait) service example <a href="#">1401</a>
BPX4SYA (symlinkat) service example <a href="#">1392</a>	BPX4WLM (__WLM) service example <a href="#">1402</a>
BPX4SYC (sysconf) service example <a href="#">1393</a>	BPX4WRT (write) service example <a href="#">1402</a>

BPX4WRV (writev) service  
     example [1402](#)  
 BPX4WTE (wait extension) service  
     example [1403](#)  
 BPX4xxx module [8](#)  
 BPXFLK, BPX4FLK (flock service [225](#))  
 BPXGMCDE, BPXGMCD4 (IPCSDumpOpenClose) service [352](#)  
 BPXGMPTR, BPXGMPT4 (IPCSDumpAccess) service [346](#)  
 BPXYACC mapping macro [1055](#)  
 BPXYAIO mapping macro [1055](#), [1187](#)  
 BPXYATT mapping macro [1057](#)  
 BPXYAUDT mapping macro [1060](#)  
 BPXYBRLK mapping macro [1060](#)  
 BPXYCCA mapping macro [1060](#), [1189](#)  
 BPXYCID mapping macro [1061](#)  
 BPXYCLNP mapping macro [1062](#)  
 BPXYCONS mapping macro [1062](#)  
 BPXYCW mapping macro [1069](#)  
 BPXYDCOR mapping macro [1069](#), [1190](#)  
 BPXYDIRE mapping macro [1074](#)  
 BPXYEFD mapping macro [1075](#)  
 BPXYENFO mapping macro [1075](#)  
 BPXYEPOL mapping macro [1075](#)  
 BPXYERNO mapping macro [1076](#)  
 BPXYFCTL mapping macro [1076](#)  
 BPXYFDUM mapping macro [1077](#)  
 BPXYFLK mapping macro [1078](#)  
 BPXYFTYP mapping macro [1078](#)  
 BPXYFUIO mapping macro [1078](#)  
 BPXYGIDN mapping macro [1079](#)  
 BPXYGIDS mapping macro [1080](#)  
 BPXYGRD mapping macro [1080](#)  
 BPXYINFY mapping macro [1080](#)  
 BPXYINHE mapping macro [1084](#), [1195](#)  
 BPXYIOC6 mapping macro [1094](#), [1196](#)  
 BPXYIOCC mapping macro [1085](#)  
 BPXYIOV mapping macro [1098](#), [1199](#)  
 BPXYIPCP mapping macro [1098](#)  
 BPXYIPCQ mapping macro [1098](#), [1199](#)  
 BPXYITIM mapping macro [1101](#), [1202](#)  
 BPXYMMG mapping macro [1101](#), [1202](#)  
 BPXYMNTE mapping macro [1104](#)  
 BPXYMODE mapping macro [1106](#)  
 BPXYMSG mapping macro [1107](#), [1204](#)  
 BPXYMSGF mapping macro [1107](#)  
 BPXYMSGH mapping macro [1109](#), [1205](#)  
 BPXYMSGX mapping macro [1109](#)  
 BPXYMTM mapping macro [1110](#)  
 BPXYOCRT mapping macro [1111](#), [1206](#)  
 BPXYOEXT mapping macro [1112](#)  
 BPXYOPNF mapping macro [1113](#)  
 BPXYPCF mapping macro [1114](#)  
 BPXPEDB mapping macro [1114](#)  
 BPXYPGPS mapping macro [1116](#)  
 BPXYPGTH mapping macro [1119](#)  
 BPXYPOE mapping macro [1123](#)  
 BPXPOLL mapping macro [1124](#)  
 BPXYPPSD mapping macro [1124](#), [1206](#)  
 BPXYPRLI mapping macro [1126](#)  
 BPXYPTAT mapping macro [1127](#)  
 BPXYPTRC mapping macro [1127](#)  
 BPXYPTXL mapping macro [1141](#), [1208](#)  
 BPXYRFIS mapping macro [1141](#)  
 BPXYRLIM mapping macro [1142](#), [1208](#)

BPXYRMON mapping macro [1142](#)  
 BPXYSECI mapping macro [1144](#)  
 BPXYSECO mapping macro [1144](#)  
 BPXYSECT mapping macro [1144](#)  
 BPXYSEEK mapping macro [1144](#)  
 BPXYSEL mapping macro [1145](#)  
 BPXYSELT mapping macro [1145](#), [1209](#)  
 BPXYSEM mapping macro [1146](#), [1209](#)  
 BPXYSFDL mapping macro [1146](#)  
 BPXYSFPL mapping macro [1147](#), [1210](#)  
 BPXYSHM mapping macro [1147](#), [1210](#)  
 BPXYSIGH mapping macro [1148](#)  
 BPXYSINF mapping macro [1150](#), [1211](#)  
 BPXYSMC mapping macro [1150](#)  
 BPXYSOCK mapping macro [1151](#)  
 BPXYSSET mapping macro [1160](#), [1211](#)  
 BPXYSSTF mapping macro [1161](#)  
 BPXYSTAT mapping macro [1162](#)  
 BPXYSXIB syscall information block [1440](#)  
 BPXYSYSC [1445](#)  
 BPXYTCCP mapping macro [1163](#)  
 BPXYTHDQ mapping macro [1164](#)  
 BPXYTHLI mapping macro [1165](#)  
 BPXYTIMS mapping macro [1168](#)  
 BPXYTIOS mapping macro [1169](#)  
 BPXYUTSN mapping macro [1172](#)  
 BPXYWAST mapping macro [1172](#)  
 BPXYWLM mapping macro [1173](#), [1212](#)  
 BPXYWNSZ mapping macro [1179](#)  
 BPXZOAPB mapping macro [1179](#)  
 BPXZOCVT mapping macro [1180](#)  
 BPXZOTCB mapping macro [1181](#)  
 buffer  
     flush I/O [923](#)  
     flush terminal [923](#)  
     write to a file [1032](#)

## C

C header  
     syscalls.h [1445](#)  
 CALL macro [1](#)  
 callable service  
     \_\_console() (BPX1CCS, BPX4CCS) [142](#)  
     \_\_cpl (BPX1CPL) [146](#)  
     \_\_getthent (BPX1GTH, BPX4GTH) [325](#)  
     \_\_map\_init (BPX1MMI, BPX4MMI) [412](#)  
     \_\_map\_service (BPX1MMS, BPX4MMS) [416](#)  
     \_\_mount (BPX2MNT, BPX4MNT) [447](#)  
     \_\_passwd (BPX1PWD, BPX4PWD) [530](#)  
     \_\_pid\_affinity (BPX1PAF, BPX4PAF) [548](#)  
     \_\_poe() (BPX1POE, BPX4POE) [559](#)  
     \_\_sigactionset (BPX1SA2, BPX4SA2) [839](#)  
     \_\_wlm (BPX1WLM, BPX4WLM) [1018](#)  
     \_authenticate (BPX1AUT, BPX4AUT)) [69](#)  
     \_exit (BPX1EXI, BPX4EXI) [180](#)  
     accept (BPX1ACP, BPX4ACP) [13](#)  
     accept\_and\_recv (BPX1ANR, BPX4ANR) [19](#)  
     Accept4\_Init (BPX1AC4, BPX4AC4)) [16](#)  
     access (BPX1ACC, BPX4ACC) [24](#)  
     accessing a [1](#)  
     aio\_suspend (BPX1ASP, BPX4ASP) [27](#)  
     alarm (BPX1ALR, BPX4ALR) [30](#)  
     asyncio (BPX1AIO, BPX4AIO) [32](#)

callable service (*continued*)

attach\_exec (BPX1ATX, BPX4ATX) [48](#)  
 attach\_execmvs (BPX1ATM, BPX4ATM) [58](#)  
 auth\_check\_resource\_np (BPX1ACK, BPX4ACK) [64](#)  
 bind (BPX1BND, BPX4BND) [76](#)  
 bind2addrsel (BPX1BAS, BPX4BAS) [79](#)  
 BPX1ACC, BPX4ACC (access) [24](#)  
 BPX1ACK, BPX4ACK (auth\_check\_resource\_np) [64](#)  
 BPX1ACP, BPX4ACP (accept) [13](#)  
 BPX1AIO, BPX4AIO (asyncio) [32](#)  
 BPX1ALR, BPX4ALR (alarm) [30](#)  
 BPX1ANR, BPX4ANR (accept\_and\_recv) [19](#)  
 BPX1ASP, BPX4ASP (aio\_suspend) [27](#)  
 BPX1ATM, BPX4ATM (attach\_execmvs) [58](#)  
 BPX1ATX, BPX4ATX (attach\_exec) [48](#)  
 BPX1BAS, BPX4BAS (bind2addrsel) [79](#)  
 BPX1BND, BPX4BND (bind) [76](#)  
 BPX1CCA, BPX4CCA (cond\_cancel) [125](#)  
 BPX1CCS, BPX4CCS (\_\_\_console()) [142](#)  
 BPX1CHA, BPX4CHA (chaudit) [95](#)  
 BPX1CHD, BPX4CHD (chdir) [98](#)  
 BPX1CHM, BPX4CHM (chmod) [101](#)  
 BPX1CHO, BPX4CHO (chown) [104](#)  
 BPX1CHP, BPX4CHP (chpriority) [107](#)  
 BPX1CHR, BPX4CHR (chattr) [82](#)  
 BPX1CLD, BPX4CLD (closedir) [124](#)  
 BPX1CLO, BPX4CLO (close) [121](#)  
 BPX1CMA, BPX4CMA (fchmodat) [206](#)  
 BPX1COA, BPX4COA (fchownat) [213](#)  
 BPX1CON, BPX4CON (connect) [139](#)  
 BPX1CPL (\_\_\_cpl) [146](#)  
 BPX1CPO, BPX4CPO (cond\_post) [127](#)  
 BPX1CRA, BPX4CRA (chatrat) [91](#)  
 BPX1CRT, BPX4CRT (chroot) [111](#)  
 BPX1CSE, BPX4CSE (cond\_setup) [130](#)  
 BPX1CTW, BPX4CTW (cond\_timed\_wait) [132](#)  
 BPX1CWA, BPX4CWA (cond\_wait) [136](#)  
 BPX1DEL, BPX4DEL (deletehfs) [148](#)  
 BPX1DSD, BPX4DSD (sw\_sigdlv) [899](#)  
 BPX1ENV, BPX4ENV (oe\_env\_np) [500](#)  
 BPX1EXC, BPX4EXC (exec) [163](#)  
 BPX1EXI, BPX4EXI (\_\_\_exit) [180](#)  
 BPX1EXM, BPX4EXM (execmvs) [175](#)  
 BPX1EXT, BPX4EXT (extlink\_np) [183](#)  
 BPX1FAA, BPX4ACC (faccessat) [186](#)  
 BPX1FAI, BPX4FAI (freeaddrinfo) [236](#)  
 BPX1FCA, BPX4FCA (fchaudit) [199](#)  
 BPX1FCD, BPX4FCD (fchdir) [202](#)  
 BPX1FCM, BPX4FCM (fchmod) [204](#)  
 BPX1FCO, BPX4FCO (fchown) [210](#)  
 BPX1FCR, BPX4FCR (fchattr) [190](#)  
 BPX1FCT, BPX4FCT (fcntl) [216](#)  
 BPX1FPC, BPX4FPC (fpathconf) [233](#)  
 BPX1FRK, BPX4FRK (fork) [227](#)  
 BPX1FST, BPX4FST (fststat) [238](#)  
 BPX1FSY, BPX4FSY (fsync) [247](#)  
 BPX1FTR, BPX4FTR (ftruncate) [249](#)  
 BPX1FTV, BPX4FTV (fstatvfs) [244](#)  
 BPX1GAI, BPX4GAI (getaddrinfo) [251](#)  
 BPX1GCL, BPX4GCL (getclientid) [258](#)  
 BPX1GCW, BPX4GCW (getcwd) [260](#)  
 BPX1GEG, BPX4GEG (getegid) [263](#)  
 BPX1GEP, BPX4GEP (getpgid) [295](#)  
 BPX1GES, BPX4GES (getsid) [317](#)

callable service (*continued*)

BPX1GET, BPX4GET (w\_getipc) [991](#)  
 BPX1GEU, BPX4GEU (geteuid) [264](#)  
 BPX1GGE, BPX4GGE (getgrent) [266](#)  
 BPX1GGI, BPX4GGI (getgrgid) [268](#)  
 BPX1GGN, BPX4GGN (getgrnam) [271](#)  
 BPX1GGR, BPX4GGR (getgroups) [274](#)  
 BPX1GHA, BPX4GHA (gethostbyaddr) [279](#)  
 BPX1GHN, BPX4GHN (gethostbyname) [282](#)  
 BPX1GID, BPX4GID (getgid) [265](#)  
 BPX1GIV, BPX4GIV (givesocket) [332](#)  
 BPX1GLG, BPX4GLG (getlogin) [289](#)  
 BPX1GMN, BPX4GMN (w\_getmntent) [995](#)  
 BPX1GNI, BPX4GNI (getnameinfo) [291](#)  
 BPX1GNM, BPX4GNM (getsockname or getpeername) [319](#)  
 BPX1GPE, BPX4GPE (getpwent) [302](#)  
 BPX1GPG, BPX4GPG (getpgrp) [297](#)  
 BPX1GPI, BPX4GPI (getpid) [298](#)  
 BPX1GPN, BPX4GPN (getpwnam) [305](#)  
 BPX1GPP, BPX4GPP (getppid) [299](#)  
 BPX1GPS (w\_getpsent) [999](#)  
 BPX1GPT, BPX4GPT (grantpt) [336](#)  
 BPX1GPU, BPX4GPU (getpwuid) [308](#)  
 BPX1GPY, BPX4GPY (getpriority) [300](#)  
 BPX1GRL, BPX4GRL (getrlimit) [313](#)  
 BPX1GRU, BPX4GRU (getrusage) [315](#)  
 BPX1GTH, BPX4GTH (\_\_\_getthent) [325](#)  
 BPX1GTR, BPX4GTR (getitimer) [287](#)  
 BPX1GUG, BPX4GUG (getgroupsbyname) [276](#)  
 BPX1GUI, BPX4GUI (getuid) [329](#)  
 BPX1GWD, BPX4GWD (getwd) [330](#)  
 BPX1HST, BPX4HST (gethostid or gethostname) [285](#)  
 BPX1IOC, BPX4IOC (w\_ioctl) [1003](#)  
 BPX1IPT, BPX4IPT (mvsipaffinity) [476](#)  
 BPX1ITY (isatty) [357](#)  
 BPX1KIL, BPX4KIL (kill) [360](#)  
 BPX1LCO, BPX4LCO (lchown) [380](#)  
 BPX1LCR, BPX4LCR (lchattr) [371](#)  
 BPX1LDX, BPX4LDX (loadhfs extended) [399](#)  
 BPX1LNK, BPX4LNK (link) [383](#)  
 BPX1LOD, BPX4LOD (loadhfs) [393](#)  
 BPX1LSK, BPX4LSK (lseek) [406](#)  
 BPX1LSN, BPX4LSN (listen) [391](#)  
 BPX1LST, BPX4LST (lstat) [409](#)  
 BPX1MAT, BPX4MAT (shmat) [804](#)  
 BPX1MCT, BPX4MCT (shmctl) [809](#)  
 BPX1MDA, BPX4MKD (mkdirat) [423](#)  
 BPX1MDT, BPX4MDT (shmdt) [812](#)  
 BPX1MGT, BPX4MGT (shmget) [827](#)  
 BPX1MKD, BPX4MKD (mkdir) [420](#)  
 BPX1MKN, BPX4MKN (mknod) [427](#)  
 BPX1MMI, BPX4MMI (\_\_\_map\_init) [412](#)  
 BPX1MMP, BPX4MMP (mmap) [435](#)  
 BPX1MMS, BPX4MMS (\_\_\_map\_service) [416](#)  
 BPX1MNA, BPX4MNA (mknodat) [431](#)  
 BPX1MNT (mount) [444](#)  
 BPX1MP, BPX4MP (mvspause) [479](#)  
 BPX1MPC, BPX4MPC (mvspocclp) [484](#)  
 BPX1MPI, BPX4MPI (mvspauseinit) [481](#)  
 BPX1MPR, BPX4MPR (mprotect) [452](#)  
 BPX1MSD, BPX4MSD (mvssigsetup) [495](#)  
 BPX1MSS, BPX4MSS (mvssigsetup) [487](#)  
 BPX1MSY, BPX4MSY (msync) [469](#)

callable service (*continued*)

BPX1MUN, BPX4MUN (munmap) [473](#)  
 BPX1NIC, BPX4NIC (nice) [497](#)  
 BPX1OAT, BPX4OAT (openat/openat2) [518](#)  
 BPX1OPD, BPX4OPD (opendir) [523](#)  
 BPX1OPN, BPX4OPN (open) [511](#)  
 BPX1OPT, BPX4OPT (getsockopt or setsockopt) [322](#)  
 BPX1PAF, BPX4PAF (\_\_\_pid\_affinity) [548](#)  
 BPX1PAS, BPX4PAS (pause) [539](#)  
 BPX1PCF, BPX4PCF (pathconf) [535](#)  
 BPX1PCT, BPX4PCT (pfsctl) [541](#)  
 BPX1PIO, BPX4PIO (w\_pioctl) [1025](#)  
 BPX1PIP, BPX4PIP (pipe) [551](#)  
 BPX1POE, BPX4POE (\_\_\_poe()) [559](#)  
 BPX1POL, BPX4POL (poll) [564](#)  
 BPX1PRL, BPX4PRL (prlimit) [574](#)  
 BPX1PSI, BPX4PSI (pthread\_setinrttype) [610](#)  
 BPX1PST, BPX4PST (pthread\_setinrttype) [613](#)  
 BPX1PTB, BPX4PTB (pthread\_cancel) [579](#)  
 BPX1PTC, BPX4PTC (pthread\_create) [581](#)  
 BPX1PTD, BPX4PTD (pthread\_detach) [586](#)  
 BPX1PTI, BPX4PTI (pthread\_testintr) [618](#)  
 BPX1PTJ, BPX4PTJ (pthread\_join) [592](#)  
 BPX1PTK, BPX4PTK (pthread\_kill) [595](#)  
 BPX1PTQ, BPX4PTQ (pthread\_quiesce) [598](#)  
 BPX1PTR, BPX4PTR (ptrace) [620](#)  
 BPX1PTS, BPX4PTS (pthread\_self) [609](#)  
 BPX1PTT, BPX4PTT (pthread\_tag\_np) [615](#)  
 BPX1PTX, BPX4PTX (pthread\_exit\_and\_get) [588](#)  
 BPX1PWD, BPX4PWD (\_\_\_passwd) [530](#)  
 BPX1QCT, BPX4QCT (msgctl) [455](#)  
 BPX1QDB, BPX4QDB (querydub) [646](#)  
 BPX1QGT, BPX4QGT (msgget) [458](#)  
 BPX1QRC, BPX4QRC (msgrcv) [462](#)  
 BPX1QSE, BPX4QSE (quiesce) [650](#)  
 BPX1QSN, BPX4QSN (msgsnd) [465](#)  
 BPX1RCV, BPX4RCV (recv) [680](#)  
 BPX1RD2, BPX4RD2 (readdir2) [659](#)  
 BPX1RDD, BPX4RDD (readdir) [656](#)  
 BPX1RDL, BPX4RDL (readlink) [666](#)  
 BPX1RDV, BPX4RDV (readv) [673](#)  
 BPX1RDX, BPX4RDX (read\_extlink) [663](#)  
 BPX1RED, BPX4RED (read) [652](#)  
 BPX1REN, BPX4REN (rename) [690](#)  
 BPX1RFM, BPX4RFM (recvfrom) [683](#)  
 BPX1RLA, BPX4RDL (readlink) [669](#)  
 BPX1RMD, BPX4RMD (rmdir) [703](#)  
 BPX1RMG, BPX4RMG (resource) [699](#)  
 BPX1RPH, BPX4RPH (realpath) [677](#)  
 BPX1RW, BPX4RW (pread() and pwrite()) [572](#)  
 BPX1RWD, BPX4RWD (rewinddir) [701](#)  
 BPX1SA2, BPX4SA2 (\_\_\_sigactionset) [839](#)  
 BPX1SCT, BPX4SCT (semctl) [713](#)  
 BPX1SDD, BPX4SDD (set\_dub\_default) [752](#)  
 BPX1SEC, BPX4SEC [364](#)  
 BPX1SEG, BPX4SEG (setegid) [758](#)  
 BPX1SEL, BPX4SEL (select) [706](#)  
 BPX1SEU, BPX4SEU (seteuid) [760](#)  
 BPX1SF, BPX4SF (send\_file) [730](#)  
 BPX1SGE, BPX4SGE (setgrent) [765](#)  
 BPX1SGI, BPX4SGI (setgid) [762](#)  
 BPX1SGQ, BPX4SGQ (sigqueue) [848](#)  
 BPX1SGR, BPX4SGR (setgroups) [766](#)  
 BPX1SGT, BPX4SGT (semget) [718](#)

callable service (*continued*)

BPX1SHT, BPX4SHT (shutdown) [832](#)  
 BPX1SIA, BPX4SIA (sigaction) [834](#)  
 BPX1SIN, BPX4SIN (server\_init) [742](#)  
 BPX1SIP, BPX4SIP (sigpending) [843](#)  
 BPX1SLK, BPX4SLK (shmем\_lock) [814](#)  
 BPX1SLP, BPX4SLP (sleep) [859](#)  
 BPX1SMC, BPX4SMC (shmем\_mutex\_condvar) [819](#)  
 BPX1SMF, BPX4SMF (smf\_record) [861](#)  
 BPX1SND, BPX4SND (send) [726](#)  
 BPX1SOC, BPX4SOC (socket or socketpair) [865](#)  
 BPX1SOP, BPX4SOP (semop) [723](#)  
 BPX1SPB, BPX4SPB (queue\_interrupt) [648](#)  
 BPX1SPE, BPX4SPE (setpwent) [782](#)  
 BPX1SPG, BPX4SPG (setpgid) [777](#)  
 BPX1SPM, BPX4SPM (sigprocmask) [845](#)  
 BPX1SPN, BPX4SPN (spawn) [869](#)  
 BPX1SPR, BPX4SPR (setpeer) [775](#)  
 BPX1SPW, BPX4SPW (server\_pwu) [746](#)  
 BPX1SPY, BPX4SPY (setpriority) [779](#)  
 BPX1SRG, BPX4SRG (setregid) [784](#)  
 BPX1SRL, BPX4SRL (setrlimit) [789](#)  
 BPX1SRU, BPX4SRU (setreuid) [786](#)  
 BPX1SRX, BPX4SRX (srx\_np) [886](#)  
 BPX1SSI, BPX4SSI (setsid) [793](#)  
 BPX1SSU, BPX4SSU (sigsuspend) [851](#)  
 BPX1STA, BPX4STA (stat) [893](#)  
 BPX1STE, BPX4STE (set\_timer\_event) [798](#)  
 BPX1STF, BPX4STF (w\_statvfs) [1029](#)  
 BPX1STL, BPX4STL (set\_thread\_limits) [795](#)  
 BPX1STO, BPX4STO (sendto) [738](#)  
 BPX1STR, BPX4STR (setitimer) [768](#)  
 BPX1STV, BPX4STV (statvfs) [896](#)  
 BPX1STW, BPX4STW (sigtimedwait) [854](#)  
 BPX1SUI, BPX4SUI (setuid) [800](#)  
 BPX1SWT, BPX4SWT (sigwait) [857](#)  
 BPX1SYA, BPX4SYA (symlinkat) [905](#)  
 BPX1SYC, BPX4SYC (sysconf) [913](#)  
 BPX1SYF, BPX4SYF (syncfs) [912](#)  
 BPX1SYM, BPX4SYM (symlink) [900](#)  
 BPX1SYN, BPX4SYN (sync) [910](#)  
 BPX1TAF, BPX4TAF (MVSThreadAffinity) [492](#)  
 BPX1TAK, BPX4TAK (takesocket) [916](#)  
 BPX1TDR, BPX4TDR (tcdrain) [918](#)  
 BPX1TFH, BPX4TFH (tcflush) [923](#)  
 BPX1TFW, BPX4TFW (tcflow) [921](#)  
 BPX1TGA, BPX4TGA (tcgetattr) [926](#)  
 BPX1TGC, BPX4TGC (tcgetcp) [928](#)  
 BPX1TGP, BPX4TGP (tcgetpgrp) [931](#)  
 BPX1TGS, BPX4TGS (tcgetsid) [933](#)  
 BPX1TIM, BPX4TIM (times) [950](#)  
 BPX1TLS, BPX4TLS (pthread\_security\_np) [601](#)  
 BPX1TRU, BPX4TRU (truncate) [953](#)  
 BPX1TSA, BPX4TSA (tcsetattr) [937](#)  
 BPX1TSB, BPX4TSB (tcsetbreak) [934](#)  
 BPX1TSP, BPX4TSP (tcsetpgrp) [944](#)  
 BPX1TST, BPX4TST (tcsettables) [946](#)  
 BPX1TYN, BPX4TYN (ttyname) [956](#)  
 BPX1ULA, BPX4ULA (unlink) [969](#)  
 BPX1UMK, BPX4UMK (umask) [960](#)  
 BPX1UMT, BPX4UMT (umount) [961](#)  
 BPX1UNA, BPX4UNA (uname) [964](#)  
 BPX1UNL, BPX4UNL (unlink) [966](#)  
 BPX1UPT, BPX4UPT (unlockpt) [973](#)



callable service (*continued*)

BPX1UQS, BPX4UQS (unquiesce) [975](#)  
 BPX1UTI, BPX4UTI (utime) [981](#)  
 BPX1WAT, BPX4WAT (wait) [984](#)  
 BPX1WLM, BPX4WLM (`_wlm`) [1018](#)  
 BPX1WRT, BPX4WRT (write) [1032](#)  
 BPX1WRV, BPX4WRV (writev) [1036](#)  
 BPX1WTE, BPX4WTE (wait-extension) [987](#)  
 BPX2ITY, BPX4ITY (isatty) [359](#)  
 BPX2MNT, BPX4MNT (`_mount`) [447](#)  
 BPX2OPN, BPX4OPS (openstat) [526](#)  
 BPX2RMS, BPX4RMS (recvmsg) [687](#)  
 BPX2SMS, BPX4SMS (sendmsg) [734](#)  
 BPX2TYN, BPX4TYN (ttyname) [958](#)  
 BPXGMPTR, BPXGMPT4 (IPCSDumpAccess) [346](#)  
 chattr (BPX1CHR, BPX4CHR) [82](#)  
 chatrat (BPX1CRA, BPX4CRA) [91](#)  
 chaudit (BPX1CHA, BPX4CHA) [95](#)  
 chdir (BPX1CHD, BPX4CHD) [98](#)  
 chmod (BPX1CHM, BPX4CHM) [101](#)  
 chown (BPX1CHO, BPX4CHO) [104](#)  
 chpriority (BPX1CHP, BPX4CHP) [107](#)  
 chroot (BPX1CRT, BPX4CRT) [111](#)  
 clone (BPX1CLN, BPX4CLN) [114](#)  
 close (BPX1CLO, BPX4CLO) [121](#)  
 closedir (BPX1CLD, BPX4CLD) [124](#)  
 cond\_cancel (BPX1CCA, BPX4CCA) [125](#)  
 cond\_post (BPX1CPO, BPX4CPO) [127](#)  
 cond\_setup (BPX1CSE, BPX4CSE) [130](#)  
 cond\_timed\_wait (BPX1CTW, BPX4CTW) [132](#)  
 cond\_wait (BPX1CWA, BPX4CWA) [136](#)  
 connect (BPX1CON, BPX4CON) [139](#)  
 deletehfs (BPX1DEL, BPX4DEL) [148](#)  
 epoll\_create (BPX1EPC, BPX4EPC) [150](#)  
 epoll\_ctl (BPX1EPL, BPX4EPL) [152](#)  
 epoll\_wait (BPX1EPW, BPX4EPW) [157](#)  
 eventfd (BPX1EFD, BPX4EFD) [160](#)  
 exec (BPX1EXC, BPX4EXC) [163](#)  
 execmvs (BPX1EXM, BPX4EXM) [175](#)  
 exit [180](#)  
 extlink\_np (BPX1EXT, BPX4EXT) [183](#)  
 faccessat (BPX1FAA, BPX4ACC) [186](#)  
 fchattr (BPX1FCR, BPX4FCR) [190](#)  
 fchaudit (BPX1FCA, BPX4FCA) [199](#)  
 fchdir (BPX1FCD, BPX4FCD) [202](#)  
 fchmod (BPX1FCM, BPX4FCM) [204](#)  
 fchmodat (BPX1CMA, BPX4CMA) [206](#)  
 fchown (BPX1FCO, BPX4FCO) [210](#)  
 fchownat (BPX1COA, BPX4COA) [213](#)  
 fcntl (BPX1FCT, BPX4FCT) [216](#)  
 flock (BPX1FLK, BPX4FLK) [225](#)  
 fork (BPX1FRK, BPX4FRK) [227](#)  
 fpathconf (BPX1FPC, BPX4FPC) [233](#)  
 freeaddrinfo (BPX1FAI, BPX4FAI) [236](#)  
 fstat (BPX1FST, BPX4FST) [238](#)  
 fstatat (BPX1FSA, BPX4FSA) [241](#)  
 fstatvfs (BPX1FTV, BPX4FTV) [244](#)  
 fsync (BPX1FSY, BPX4FSY) [247](#)  
 ftruncate (BPX1FTR, BPX4FTR) [249](#)  
 getaddrinfo (BPX1GAI, BPX4GAI) [251](#)  
 getclientid (BPX1GCL, BPX4GCL) [258](#)  
 getcwd (BPX1GCW, BPX4GCW) [260](#)  
 getegid (BPX1GEG, BPX4GEG) [263](#)  
 geteuid (BPX1GEU, BPX4GEU) [264](#)

callable service (*continued*)

getgid (BPX1GID, BPX4GID) [265](#)  
 getgrent (BPX1GGE, BPX4GGE) [266](#)  
 getgrgid (BPX1GGI, BPX4GGI) [268](#)  
 getgrnam (BPX1GGN, BPX4GGN) [271](#)  
 getgroups (BPX1GGR, BPX4GGR) [274](#)  
 getgroupsbyname (BPX1GUG, BPX4GUG) [276](#)  
 gethostbyaddr (BPX1GHA, BPX4GHA) [279](#)  
 gethostbyname (BPX1GHN, BPX4GHN) [282](#)  
 gethostid or gethostname (BPX1HST, BPX4HST) [285](#)  
 getitimer (BPX1GTR, BPX4GTR) [287](#)  
 getlogin (BPX1GLG, BPX4GLG) [289](#)  
 getnameinfo (BPX1GNI, BPX4GNI) [291](#)  
 getpgid (BPX1GEP, BPX4GEP) [295](#)  
 getpgrp (BPX1GPG, BPX4GPG) [297](#)  
 getpid (BPX1GPI, BPX4GPI) [298](#)  
 getppid (BPX1GPP, BPX4GPP) [299](#)  
 getpriority (BPX1GPY, BPX4GPY) [300](#)  
 getpwent (BPX1GPE, BPX4GPE) [302](#)  
 getpwnam (BPX1GPN, BPX4GPN) [305](#)  
 getpwuid (BPX1GPU, BPX4GPU) [308](#)  
 getrandom (BPX1GRD, BPX4GRD) [310](#)  
 getrlimit (BPX1GRL, BPX4GRL) [313](#)  
 getrusage (BPX1GRU, BPX4GRU) [315](#)  
 getsid (BPX1GES, BPX4GES) [317](#)  
 getsockname or getpeername (BPX1GNM, BPX4GNM) [319](#)  
 getsockopt or setsockopt (BPX1OPT, BPX4OPT) [322](#)  
 getuid (BPX1GUI, BPX4GUI) [329](#)  
 getwd (BPX1GWD, BPX4GWD) [330](#)  
 givesocket (BPX1GIV, BPX4GIV) [332](#)  
 grantpt (BPX1GPT, BPX4GPT) [336](#)  
 inotify\_add\_watch (BPX1IAW, BPX4IAW) [338](#)  
 inotify\_init (BPX1INI, BPX4INI) [341](#)  
 inotify\_rm\_watch (BPX1IRW, BPX4IRW) [344](#)  
 IPCSDumpAccess (BPXGMPTR, BPXGMPT4) [346](#)  
 isatty (BPX1ITY) [357](#)  
 isatty (BPX2ITY, BPX4ITY) [359](#)  
 kill (BPX1KIL, BPX4KIL) [360](#)  
 lchattr (BPX1LCR, BPX4LCR) [371](#)  
 lchown (BPX1LCO, BPX4LCO) [380](#)  
 link (BPX1LNK, BPX4LNK) [383](#)  
 linkat (BPX1LNA, BPX4LNA) [387](#)  
 listen (BPX1LSN, BPX4LSN) [391](#)  
 loadhfs (BPX1LOD, BPX4LOD) [393](#)  
 loadhfs extended (BPX1LDX, BPX4LDX) [399](#)  
 lseek (BPX1LSK, BPX4LSK) [406](#)  
 lstat (BPX1LST, BPX4LST) [409](#)  
 mkdir (BPX1MKD, BPX4MKD) [420](#)  
 mkdirat (BPX1MDA, BPX4MKD) [423](#)  
 mknod (BPX1MKN, BPX4MKN) [427](#)  
 mknodat (BPX1MNA, BPX4MNA) [431](#)  
 mmap (BPX1MMP, BPX4MMP) [435](#)  
 mount (BPX1MNT) [444](#)  
 mprotect (BPX1MPR, BPX4MPR) [452](#)  
 msgctl (BPX1QCT, BPX4QCT) [455](#)  
 msgget (BPX1QGT, BPX4QGT) [458](#)  
 msgrcv (BPX1QRC, BPX4QRC) [462](#)  
 msgsnd (BPX1QSN, BPX4QSN) [465](#)  
 msync (BPX1MSY, BPX4MSY) [469](#)  
 munmap (BPX1MUN, BPX4MUN) [473](#)  
 mvsiptaffinity (BPX1IPT, BPX4IPT) [476](#)  
 mvspause (BPX1MP, BPX4MP) [479](#)  
 mvspauseinit (BPX1MPI, BPX4MPI) [481](#)

callable service (*continued*)

[mvspocclp \(BPX1MPC, BPX4MPC\) 484](#)  
[mvssigsetup \(BPX1MSS, BPX4MSS\) 487](#)  
[MVSThreadAffinity \(BPX1TAF, BPX4TAF\) 492](#)  
[mvvsunsigsetup \(BPX1MSD, BPX4MSD\) 495](#)  
[nice \(BPX1NIC, BPX4NIC\) 497](#)  
[oe\\_env\\_np \(BPX1ENV, BPX4ENV\) 500](#)  
[open \(BPX1OPN, BPX4OPN\) 511](#)  
[openat/openat2 \(BPX1OAT, BPX4OAT\) 518](#)  
[opendir \(BPX1OPD, BPX4OPD\) 523](#)  
[openstat \(BPX2OPN, BPX4OPS\) 526](#)  
[pathconf \(BPX1PCF, BPX4PCF\) 535](#)  
[pause \(BPX1PAS, BPX4PAS\) 539](#)  
[pfscctl \(BPX1PCT, BPX4PCT\) 541](#)  
[pipe \(BPX1PIP, BPX4PIP\) 551](#)  
[pipe2 \(BPX1PI2, BPX4PI2\) 553](#)  
[pivot\\_root \(BPX1PVR, BPX4PVR\) 556](#)  
[poll \(BPX1POL, BPX4POL\) 564](#)  
[prctl \(BPX1PRC, BPX4PRC\) 568](#)  
[pread\(\) and pwrite\(\) \(BPX1RW, BPX4RW\) 572](#)  
[prlimit \(BPX1PRL, BPX4PRL\) 574](#)  
[pthread\\_cancel \(BPX1PTB, BPX4PTB\) 579](#)  
[pthread\\_create \(BPX1PTC, BPX4PTC\) 581](#)  
[pthread\\_detach \(BPX1PTD, BPX4PTD\) 586](#)  
[pthread\\_exit\\_and\\_get \(BPX1PTX, BPX4PTX\) 588](#)  
[pthread\\_join \(BPX1PTJ, BPX4PTJ\) 592](#)  
[pthread\\_kill \(BPX1PTK, BPX4PTK\) 595](#)  
[pthread\\_quiesce \(BPX1PTQ, BPX4PTQ\) 598](#)  
[pthread\\_security\\_np \(BPX1TLS, BPX4TLS\) 601](#)  
[pthread\\_self \(BPX1PTS, BPX4PTS\) 609](#)  
[pthread\\_setintr \(BPX1PSI, BPX4PSI\) 610](#)  
[pthread\\_setintrtype \(BPX1PST, BPX4PST\) 613](#)  
[pthread\\_tag\\_np \(BPX1PTT, BPX4PTT\) 615](#)  
[pthread\\_testintr \(BPX1PTI, BPX4PTI\) 618](#)  
[ptrace \(BPX1PTR, BPX4PTR\) 620](#)  
[querydub \(BPX1QDB, BPX4QDB\) 646](#)  
[queue\\_interrupt \(BPX1SPB, BPX4SPB\) 648](#)  
[quiesce \(BPX1QSE, BPX4QSE\) 650](#)  
[read \(BPX1RED, BPX4RED\) 652](#)  
[read\\_extlink \(BPX1RDX, BPX4RDX\) 663](#)  
[readdir \(BPX1RDD, BPX4RDD\) 656](#)  
[readdir2 \(BPX1RD2, BPX4RD2\) 659](#)  
[readlink \(BPX1RDL, BPX4RDL\) 666](#)  
[readlink \(BPX1RLA, BPX4RDL\) 669](#)  
[readv \(BPX1RDV, BPX4RDV\) 673](#)  
[realpath \(BPX1RPH, BPX4RPH\) 677](#)  
[recv \(BPX1RCV, BPX4RCV\) 680](#)  
[recvfrom \(BPX1RFM, BPX4RFM\) 683](#)  
[recvmmsg \(BPX2RMS, BPX4RMS\) 687](#)  
[rename \(BPX1REN, BPX4REN\) 690](#)  
[renameat \(BPX1RNA, BPX4RNA\) 694](#)  
[resource \(BPX1RMG, BPX4RMG\) 699](#)  
[rewinddir \(BPX1RWD, BPX4RWD\) 701](#)  
[rmdir \(BPX1RMD, BPX4RMD\) 703](#)  
[select \(BPX1SEL, BPX4SEL\) 706](#)  
[semctl \(BPX1SCT, BPX4SCT\) 713](#)  
[semget \(BPX1SGT, BPX4SGT\) 718](#)  
[semop \(BPX1SOP, BPX4SOP\) 723](#)  
[send \(BPX1SND, BPX4SND\) 726](#)  
[send\\_file \(BPX1SF, BPX4SF\) 730](#)  
[sendmsg \(BPX2SMS, BPX4SMS\) 734](#)  
[sendto \(BPX1STO, BPX4STO\) 738](#)  
[server\\_init \(BPX1SIN, BPX4SIN\) 742](#)  
[server\\_pwu \(BPX1SPW, BPX4SPW\) 746](#)

callable service (*continued*)

[set\\_dub\\_default \(BPX1SDD, BPX4SDD\) 752](#)  
[set\\_thread\\_limits \(BPX1STL, BPX4STL\) 795](#)  
[set\\_timer\\_event \(BPX1STE, BPX4STE\) 798](#)  
[setegid \(BPX1SEG, BPX4SEG\) 758](#)  
[seteuid \(BPX1SEU, BPX4SEU\) 760](#)  
[setgid \(BPX1SGI, BPX4SGI\) 762](#)  
[setgrent \(BPX1SGE, BPX4SGE\) 765](#)  
[setgroups \(BPX1SGR, BPX4SGR\) 766](#)  
[setitimer \(BPX1STR, BPX4STR\) 768](#)  
[setns \(BPX1SNS, BPX4SNS\) 771](#)  
[setpeer \(BPX1SPR, BPX4SPR\) 775](#)  
[setpgid \(BPX1SPG, BPX4SPG\) 777](#)  
[setpriority \(BPX1SPY, BPX4SPY\) 779](#)  
[setpwent \(BPX1SPE, BPX4SPE\) 782](#)  
[setregid \(BPX1SRG, BPX4SRG\) 784](#)  
[setreuid \(BPX1SRU, BPX4SRU\) 786](#)  
[setrlimit \(BPX1SRL, BPX4SRL\) 789](#)  
[setsid \(BPX1SSI, BPX4SSI\) 793](#)  
[setuid \(BPX1SUI, BPX4SUI\) 800](#)  
[shmat \(BPX1MAT, BPX4MAT\) 804](#)  
[shmctl \(BPX1MCT, BPX4MCT\) 809](#)  
[shmdt \(BPX1MDT, BPX4MDT\) 812](#)  
[shmем\\_lock \(BPX1SLK, BPX4SLK\) 814](#)  
[shmем\\_mutex\\_condvar \(BPX1SMC, BPX4SMC\) 819](#)  
[shmget \(BPX1MGT, BPX4MGT\) 827](#)  
[shutdown \(BPX1SHT, BPX4SHT\) 832](#)  
[sigaction \(BPX1SIA, BPX4SIA\) 834](#)  
[sigpending \(BPX1SIP, BPX4SIP\) 843](#)  
[sigprocmask \(BPX1SPM, BPX4SPM\) 845](#)  
[sigqueue \(BPX1SGQ, BPX4SGQ\) 848](#)  
[sigsuspend \(BPX1SSU, BPX4SSU\) 851](#)  
[sigtimedwait \(BPX1STW, BPX4STW\) 854](#)  
[sigwait \(BPX1SWT, BPX4SWT\) 857](#)  
[sleep \(BPX1SLP, BPX4SLP\) 859](#)  
[smf\\_record \(BPX1SMF, BPX4SMF\) 861](#)  
[socket or socketpair \(BPX1SOC, BPX4SOC\) 865](#)  
[spawn \(BPX1SPN, BPX4SPN\) 869](#)  
[srx\\_np \(BPX1SRX, BPX4SRX\) 886](#)  
[stat \(BPX1STA, BPX4STA\) 893](#)  
[statvfs \(BPX1STV, BPX4STV\) 896](#)  
[sw\\_sigdlv \(BPX1DSD, BPX4DSD\) 899](#)  
[symlink \(BPX1SYM, BPX4SYM\) 900](#)  
[symlinkat \(BPX1SYA, BPX4SYA\) 905](#)  
[sync \(BPX1SYN, BPX4SYN\) 910](#)  
[syncfs \(BPX1SYF, BPX4SYF\) 912](#)  
[syntax 1](#)  
[sysconf \(BPX1SYC, BPX4SYC\) 913](#)  
[takesocket \(BPX1TAK, BPX4TAK\) 916](#)  
[tcdrain \(BPX1TDR, BPX4TDR\) 918](#)  
[tcflow \(BPX1TFW, BPX4TFW\) 921](#)  
[tcflush \(BPX1TFH, BPX4TFH\) 923](#)  
[tcgetattr \(BPX1TGA, BPX4TGA\) 926](#)  
[tcgetcp \(BPX1TGC, BPX4TGC\) 928](#)  
[tcgetpgrp \(BPX1TGP, BPX4TGP\) 931](#)  
[tcgetsid \(BPX1TGS, BPX4TGS\) 933](#)  
[tcsendbreak \(BPX1TSB, BPX4TSB\) 934](#)  
[tcsetattr \(BPX1TSA, BPX4TSA\) 937](#)  
[tcsetpgrp \(BPX1TSP, BPX4TSP\) 944](#)  
[tcsettables \(BPX1TST, BPX4TST\) 946](#)  
[times \(BPX1TIM, BPX4TIM\) 950](#)  
[truncate \(BPX1TRU, BPX4TRU\) 953](#)  
[ttyname \(BPX1TYN, BPX4TYN\) 956](#)  
[ttyname \(BPX2TYN, BPX4TYN\) 958](#)

callable service (*continued*)

- umask (BPX1UMK, BPX4UMK) [960](#)
- umount (BPX1UMT, BPX4UMT) [961](#)
- uname (BPX1UNA, BPX4UNA) [964](#)
- unlink (BPX1UNL, BPX4UNL) [966](#)
- unlinkat (BPX1ULA, BPX4ULA) [969](#)
- unlockpt (BPX1UPT, BPX4UPT) [973](#)
- unquiesce (BPX1UQS, BPX4UQS) [975](#)
- unshare (BPX1UNS, BPX4UNS) [977](#)
- utime (BPX1UTI, BPX4UTI) [981](#)
- w\_getipc (BPX1GET, BPX4GET) [991](#)
- w\_getmntent (BPX1GMN, BPX4GMN) [995](#)
- w\_getpsent (BPX1GPS) [999](#)
- w\_ioctl (BPX1IOC, BPX4IOC) [1003](#)
- w\_pioclt (BPX1PIO, BPX4PIO) [1025](#)
- w\_statvfs (BPX1STF, BPX4STF) [1029](#)
- wait (BPX1WAT, BPX4WAT) [984](#)
- wait-extension (BPX1WTE, BPX4WTE) [987](#)
- write (BPX1WRT, BPX4WRT) [1032](#)
- writew (BPX1WRV, BPX4WRV) [1036](#)

callable service examples

- nonreentrant [1407](#), [1409](#)

callable service numbers [1445](#)

calling process

- cancel a thread [579](#)
- create a thread [581](#)
- obtain effective group ID of [263](#)
- obtain effective user ID of [264](#)
- server initialization [742](#)
- server process work unit [746](#)
- WLM interface service [1018](#)

cancel

- interest in events [125](#)
- thread [579](#)

capturing storage in a debugged process [640](#)

certificate

- perform security-related services [364](#)

change

- audit flags for a file
  - by descriptor [199](#)
- directory
  - by descriptor [204](#)
- directory mode [101](#), [206](#)
- file mode
  - by descriptor [204](#)
- file offset [406](#)
- group of a directory
  - by descriptor [210](#)
- group of a file
  - by descriptor [210](#)
- interrupt state [610](#)
- interrupt type [613](#)
- owner of a directory
  - by descriptor [210](#)
- owner of a file
  - by descriptor [210](#)
- process's signal mask [845](#)
- root directory [111](#)
- signal action [834](#)
- signal actions [839](#)
- signal mask [851](#)
- working directory [98](#), [202](#)

chattr (BPX1CHR, BPX4CHR) service [82](#)

chattr (BPX1CHR) service (*continued*)

- example [1228](#)

chattr (BPX4CHR) service

- example [1321](#)

chatrat (BPX1CRA, BPX4CRA) [91](#)

chatrat (BPX1CRA) service

- example [1230](#)

chatrat (BPX4CRA) service

- example [1323](#)

chaudit (BPX1CHA, BPX4CHA) service [95](#)

chaudit (BPX1CHA) service

- example [1225](#)

chaudit (BPX4CHA) service

- example [1319](#)

chdir (BPX1CHD, BPX4CHD) service [98](#)

chdir (BPX1CHD) service

- example [1226](#)

chdir (BPX4CHD) service

- example [1319](#)

check

- file availability [24](#)
- user permission [186](#)

child process

- cloning [114](#)
- create [227](#)
- creating [114](#)
- obtain process time [950](#)
- status of stopped [984](#)

chmod (BPX1CHM, BPX4CHM) service [101](#)

chmod (BPX1CHM) service

- example [1226](#)

chmod (BPX4CHM) service

- example [1319](#)

chown (BPX1CHO, BPX4CHO) service [104](#)

chown (BPX1CHO) service

- example [1226](#)

chown (BPX4CHO) service

- example [1320](#)

chpriority (BPX1CHP, BPX4CHP) service [107](#)

chpriority (BPX1CHP) service

- example [1227](#)

chpriority (BPX4CHP) service

- example [1320](#)

chroot (BPX1CRT, BPX4CRT) service [111](#)

chroot (BPX1CRT) service

- example [1230](#), [1231](#)

chroot (BPX4CRT) service

- example [1324](#)

clean up

- flush I/O buffer [923](#)
- kernel resources [484](#)

clear

- terminal buffer [923](#)

clone (BPX1CLN, BPX4CLN) service [114](#)

clone (BPX1CLN) service

- example [1228](#)

clone (BPX4CLN) service

- example [1322](#)

clone service

- example [1228](#), [1322](#)

cloning

- child process [114](#)

close

- directory [124](#)



- close (*continued*)
  - dump [352](#)
  - file [121](#)
- close (BPX1CLO, BPX4CLO) service [121](#)
- close (BPX1CLO) service
  - example [1229](#)
- close (BPX4CLO) service
  - example [1323](#)
- closedir (BPX1CLD, BPX4CLD) service [124](#)
- closedir (BPX1CLD) service
  - example [1229](#)
- closedir (BPX4CLD) service
  - example [1322](#)
- code page
  - get terminal [928](#)
  - set terminal [940](#)
- code page names and conversion tables
  - set terminal [946](#)
- cond\_cancel (BPX1CCA, BPX4CCA) service [125](#)
- cond\_cancel (BPX1CCA) service
  - example [1225](#)
- cond\_cancel (BPX4CCA) service
  - example [1318](#)
- cond\_post (BPX1CPO, BPX4CPO) service [127](#)
- cond\_post (BPX1CPO) service
  - example [1230](#)
- cond\_post (BPX4CPO) service
  - example [1323](#)
- cond\_setup (BPX1CSE, BPX4CSE) service [130](#)
- cond\_setup (BPX1CSE) service
  - example [1231](#)
- cond\_setup (BPX4CSE) service
  - example [1324](#)
- cond\_timed\_wait (BPX1CTW, BPX4CTW) service [132](#)
- cond\_wait (BPX1CTW) service
  - example [1231](#)
- cond\_wait (BPX1CWA, BPX4CWA) service [136](#)
- cond\_wait (BPX1CWA) service
  - example [1232](#)
- cond\_wait (BPX4CTW) service
  - example [1325](#)
- cond\_wait (BPX4CWA) service
  - example [1325](#)
- configuration
  - determine
    - limit [233](#), [535](#)
    - path name variable [233](#), [535](#)
  - system options [913](#)
- connect (BPX1CON, BPX4CON) service [139](#)
- connect (BPX1CON) service
  - example [1229](#)
- connect (BPX4CON) service
  - example [1323](#)
- contact
  - z/OS [1451](#)
- control
  - automatic conversion [216](#)
  - file descriptors [216](#)
- control I/O [1003](#), [1025](#)
- coupling facility
  - calculating structure sizes [146](#)
- create
  - character special file [427](#)
  - directory [420](#)

- create (*continued*)
  - directory relative to a directory file descriptor [423](#)
  - FIFO file [427](#)
  - link to a file [383](#)
  - mapped megabyte area [412](#)
  - multiple threads [1421](#)
  - pipe [551](#)
  - pipe instance [16](#), [553](#)
  - pthreads [1421](#)
  - session
    - set process group ID [793](#)
  - symbolic link relative to directory file descriptor [905](#)
  - symbolic link to external name [183](#)
  - symbolic link to path name [900](#)
  - thread [581](#)
  - threads [1421](#)
- creating
  - child process [114](#)
  - new process [227](#)
- creation mask
  - set or return file mode [960](#)
- current operating system
  - display name [964](#)

## D

- data block
  - change permissions for [416](#)
  - connect [416](#)
  - create [416](#)
  - disconnect [416](#)
  - free backing storage for [416](#)
- data flow
  - suspend or resume terminal [921](#)
- database
  - obtain user information [308](#)
  - user
    - access by user name [305](#), [530](#)
- debugger
  - controls [620](#)
- debugging
  - attaching to process [633](#)
  - capturing storage [640](#)
  - determining modules loaded [636](#)
  - ending a debugged process [641](#)
  - handling extended events [638](#)
  - handling program checks or abends [636](#)
  - manipulating data [639](#)
  - multiprocess debugging mode [641](#)
  - receiving notification of events [634](#)
  - resuming or detaching from a debugged process [640](#)
  - setting a breakpoint [639](#)
  - user area description [642](#)
  - working with threads [635](#)
- deletehfs (BPX1DEL, BPX4DEL) service [148](#)
- deleteHFS (BPX1DEL) service
  - example [1232](#)
- deleteHFS (BPX4DEL) service
  - example [1325](#)
- delivery key
  - signal [1416](#)
- detach
  - signal setup [495](#)
- determining modules loaded in a debugged process [636](#)

- directory
  - change
    - by descriptor [204](#)
  - change root [111](#)
  - change the group
    - by descriptor [210](#)
  - change the owner
    - by descriptor [210](#)
  - change working [98](#), [202](#)
  - changing attributes [91](#)
  - close [124](#)
  - create [420](#), [423](#)
  - determine
    - configurable limit [233](#), [535](#)
    - path name variable [233](#), [535](#)
  - open [523](#)
  - read entry [656](#), [659](#)
  - remove [703](#)
  - remove entry [966](#), [969](#)
  - rename [690](#)
  - reset to the beginning [701](#)
  - rewind to the beginning [701](#)
- disable
  - signal delivery [899](#)
- display
  - name of current operating system [964](#)
- dub [1](#), [1414](#)
- dub setting
  - change default [752](#)
- dubbed task [1](#)
- dump
  - close [352](#)
  - open [352](#)
  - read information [346](#)

## E

- effective group ID
  - obtain [263](#)
  - set [758](#)
- effective user ID
  - obtain [264](#)
  - set [760](#)
- enable
  - signal delivery [899](#)
- end process
  - bypass cleanup [180](#)
- ending a debugged process [641](#)
- environmental attribute
  - environment
    - attributes [500](#)
- environmental restrictions [5](#)
- epoll\_create (BPX1EPC, BPX4EPC) service [150](#)
- epoll\_create service
  - example [1233](#), [1326](#)
- epoll\_ctl (BPX1EPL, BPX4EPK) service [152](#)
- epoll\_ctl service
  - example [1327](#)
- epoll\_epl service
  - example [1233](#)
- epoll\_wait (BPX1EPW, BPX4EPW) service [157](#)
- epoll\_wait service
  - example [1234](#), [1327](#)
- ESPIE or ESTAE macro or routine

- ESPIE or ESTAE macro or routine (*continued*)
  - high-level language [1415](#)
  - signals [1415](#)
- eventfd (BPX1EFD, BPX4EFD) service [160](#)
- eventfd (BPX1EFD) service
  - example [1232](#)
- eventfd (BPX4EFD) service
  - example [1326](#)
- events
  - cancel interest [125](#)
  - wait on user events [479](#), [481](#)
- examine
  - interrupt state [610](#)
  - interrupt type [613](#)
  - pending signals [843](#)
  - process's signal mask [845](#)
  - signal action [834](#)
  - signal actions [839](#)
- examples of callable services
  - \_\_authenticate (BPX1AUT) [1223](#)
  - \_\_authenticate (BPX4AUT) [1317](#)
  - \_\_console() (BPX1CCS) [1225](#)
  - \_\_console() (BPX4CCS) [1319](#)
  - \_\_getthent (BPX1GTH) [1251](#)
  - \_\_getthent (BPX4GTH) [1344](#)
  - \_\_login (BPX1SEG) [1285](#)
  - \_\_login (BPX4SEC) [1377](#)
  - \_\_map\_init (BPX1MMI) [1261](#)
  - \_\_map\_init (BPX4MMI) [1354](#)
  - \_\_map\_service (BPX1MMS) [1262](#)
  - \_\_map\_service (BPX4MMS) [1355](#)
  - \_\_mount (BPX2MNT) [1263](#)
  - \_\_mount (BPX4MNT) [1356](#)
  - \_\_passwd (BPX1PWD) [1276](#)
  - \_\_passwd (BPX4PWD) [1368](#)
  - \_\_pid\_affinity (BPX1PAF) [1268](#)
  - \_\_pid\_affinity (BPX4PAF) [1360](#)
  - \_\_sigactionset (BPX1SA2) [1284](#)
  - \_\_sigactionset (BPX4SA2) [1376](#)
  - \_\_WLM (BPX1WLM) [1309](#)
  - \_\_WLM (BPX4WLM) [1402](#)
  - \_exit (BPX1EXI) [1235](#)
  - \_exit (BPX4EXI) [1328](#)
  - accept (BPX1ACP) [1221](#)
  - accept (BPX4ACP) [1315](#)
  - accept\_and\_recv (BPX1ANR) [1222](#)
  - accept\_and\_recv (BPX4ANR) [1316](#)
  - accept4 (BPX1AC4) [1220](#), [1221](#)
  - accept4 (BPX4AC4) [1314](#), [1315](#)
  - access (BPX1ACC) [1219](#)
  - access (BPX4ACC) [1313](#)
  - aio\_suspend (BPX1ASP) [1222](#)
  - aio\_suspend (BPX4ASP) [1316](#)
  - alarm (BPX1ALR) [1221](#)
  - alarm (BPX4ALR) [1315](#)
  - asyncio (BPX1AIO) [1221](#)
  - asyncio (BPX4AIO) [1315](#)
  - attach\_exec (BPX1ATX) [1223](#)
  - attach\_exec (BPX4ATX) [1317](#)
  - attach\_execmvs (BPX1ATM) [1222](#)
  - attach\_execmvs (BPX4ATM) [1316](#)
  - auth\_check\_resource\_np (BPX1ACK) [1220](#)
  - auth\_check\_resource\_np (BPX4ACK) [1314](#)
  - bind (BPX1BND) [1224](#)

examples of callable services (*continued*)

[bind \(BPX4BND\) 1318](#)  
[bind with source address selection \(BPX1BAS\) 1224](#)  
[bind with source address selection \(BPX4BAS\) 1318](#)  
[BPX1AC4 \(accept4\) 1220, 1221](#)  
[BPX1ACC \(access\) 1219](#)  
[BPX1ACK \(auth\\_check\\_resource\\_np\) 1220](#)  
[BPX1ACP \(accept\) 1221](#)  
[BPX1AIO \(asyncio\) 1221](#)  
[BPX1ALR \(alarm\) 1221](#)  
[BPX1ANR \(accept\\_and\\_recv\) 1222](#)  
[BPX1ASP \(aio\\_suspend\) 1222](#)  
[BPX1ATM \(attach\\_execmvs\) 1222](#)  
[BPX1ATX \(attach\\_exec\) 1223](#)  
[BPX1AUT \(\\_\\_authenticate\) 1223](#)  
[BPX1BAS \(bind with source address selection\) 1224](#)  
[BPX1BND \(bind\) 1224](#)  
[BPX1CCA \(cond\\_cancel\) 1225](#)  
[BPX1CCS \(\\_\\_console\(\)\) 1225](#)  
[BPX1CHA \(chaudit\) 1225](#)  
[BPX1CHD \(chdir\) 1226](#)  
[BPX1CHM \(chmod\) 1226](#)  
[BPX1CHO \(chown\) 1226](#)  
[BPX1CHOA \(fchownat\) 1226](#)  
[BPX1CHP \(chpriority\) 1227](#)  
[BPX1CHR \(chattr\) 1228](#)  
[BPX1CLD \(closedir\) 1229](#)  
[BPX1CLN \(clone\) 1228](#)  
[BPX1CLO \(close\) 1229](#)  
[BPX1CMA \(fchmodat\) 1227, 1321](#)  
[BPX1CON \(connect\) 1229](#)  
[BPX1CPO \(cond\\_post\) 1230](#)  
[BPX1CRA \(chatrat 1230, 1323\)](#)  
[BPX1CRT \(chroot\) 1230, 1231](#)  
[BPX1CSE \(cond\\_setup\) 1231](#)  
[BPX1CTW \(cond\\_timed\\_wait\) 1231](#)  
[BPX1CWA \(cond\\_wait\) 1232](#)  
[BPX1DEL \(deleteHFS\) 1232](#)  
[BPX1EFD \(eventfd\) 1232](#)  
[BPX1ENV \(oe\\_env\\_np\) 1233](#)  
[BPX1EPW \(epoll\\_wait\) 1234](#)  
[BPX1EXC \(exec\) 1234](#)  
[BPX1EXI \(\\_exit\) 1235](#)  
[BPX1EXM \(execmvs\) 1235](#)  
[BPX1EXT \(extlink\\_np\) 1235](#)  
[BPX1FAA \(faccessat\) 1235, 1329](#)  
[BPX1FAI \(freeaddrinfo\) 1236](#)  
[BPX1FCA \(fchaudit\) 1236](#)  
[BPX1FCD \(fchdir\) 1237](#)  
[BPX1FCM \(fchmod\) 1237](#)  
[BPX1FCO \(fchown\) 1237](#)  
[BPX1FCR \(fchattr\) 1237](#)  
[BPX1FCT \(fcntl\) 1238](#)  
[BPX1FLK \(flock\) example 1239](#)  
[BPX1FPC \(fpathconf\) 1239](#)  
[BPX1FRK \(fork\) 1239](#)  
[BPX1FSA \(fstatat\) 1333](#)  
[BPX1FST \(fstat\) 1239](#)  
[BPX1FSY \(fsync\) 1240](#)  
[BPX1FTR \(ftruncate\) 1240](#)  
[BPX1FTV \(fstatvfs\) 1240](#)  
[BPX1GAI \(getaddrinfo\) 1241](#)  
[BPX1GCL \(getclientid\) 1241](#)  
[BPX1GCW \(getcwd\) 1241](#)

examples of callable services (*continued*)

[BPX1GEG \(getegid\) 1242](#)  
[BPX1GEP \(getpgid\) 1242](#)  
[BPX1GES \(getsid\) 1242](#)  
[BPX1GET \(w\\_getipc\) 1242](#)  
[BPX1GEU \(geteuid\) 1243](#)  
[BPX1GGE \(getgrent\) 1243](#)  
[BPX1GGI \(getgrgid\) 1243](#)  
[BPX1GGN \(getgrnam\) 1244](#)  
[BPX1GGR \(getgroups\) 1244](#)  
[BPX1GHA \(gethostbyaddr\) 1244](#)  
[BPX1GHN \(gethostbyname\) 1245](#)  
[BPX1GID \(getgid\) 1245](#)  
[BPX1GIV \(givesocket\) 1245](#)  
[BPX1GLG \(getlogin\) 1246](#)  
[BPX1GMN \(w\\_getmntent\) 1246](#)  
[BPX1GNI \(getnameinfo\) 1246](#)  
[BPX1GNM \(getpeername or getsockname\) 1247](#)  
[BPX1GPE \(getpwent\) 1247](#)  
[BPX1GPG \(getpgrp\) 1247](#)  
[BPX1GPI \(getpid\) 1248](#)  
[BPX1GPN \(getpwnam\) 1248](#)  
[BPX1GPP \(getppid\) 1248](#)  
[BPX1GPS \(w\\_getpsent\) 1248](#)  
[BPX1GPT \(grantpt\) 1249](#)  
[BPX1GPU \(getpwuid\) 1249](#)  
[BPX1GPY \(getpriority\) 1249](#)  
[BPX1GRD \(getrandom\) 1250](#)  
[BPX1GRL \(getrlimit\) 1250](#)  
[BPX1GRU \(getrusage\) 1250](#)  
[BPX1GTH \(\\_\\_getthent\) 1251](#)  
[BPX1GTR \(getitimer\) 1251](#)  
[BPX1GUG \(getgroupsbyname\) 1251](#)  
[BPX1GUI \(getuid\) 1252](#)  
[BPX1GWD \(getwd\) 1252](#)  
[BPX1HST \(gethostid or gethostname\) 1252](#)  
[BPX1IAW \(inotify\\_add\\_watch\) 1252](#)  
[BPX1IEPC \(epoll\\_create\) 1233](#)  
[BPX1IEPL \(epoll\\_epl\) 1233](#)  
[BPX1INI \(inotify\\_init\) 1253, 1346](#)  
[BPX1IOC \(w\\_ioctl\) 1253](#)  
[BPX1IPT \(mvsiptaffinity\) 1253](#)  
[BPX1IRM \(inotify\\_rm\\_watch\) 1254](#)  
[BPX1ITY \(isatty\) 1254](#)  
[BPX1KIL \(kill\) 1254](#)  
[BPX1LCO \(lchown\) 1255](#)  
[BPX1LCR \(lchattr\) 1255](#)  
[BPX1LDX \(loadhfs extended\) 1255](#)  
[BPX1LNA \(linkat\) 1257](#)  
[BPX1LNK \(link\) 1258](#)  
[BPX1LOD \(loadhfs\) 1257](#)  
[BPX1LSK \(lseek\) 1258](#)  
[BPX1LSN \(listen\) 1258](#)  
[BPX1LST \(lstat\) 1258](#)  
[BPX1MAT \(shmat\) 1259](#)  
[BPX1MCT \(shmctl\) 1259](#)  
[BPX1MDA \(mkdirat\) 1259](#)  
[BPX1MDT \(shmdt\) 1260](#)  
[BPX1MGT \(shmget\) 1260](#)  
[BPX1MKD \(mkdir\) 1260](#)  
[BPX1MKN \(mknod\) 1261](#)  
[BPX1MMI \(\\_\\_map\\_init\) 1261](#)  
[BPX1MMP \(mmap\) 1262](#)  
[BPX1MMS \(\\_\\_map\\_service\) 1262](#)

examples of callable services (*continued*)

[BPX1MNA \(mknodat\) 1262](#)  
[BPX1MNT \(mount\) 1263](#)  
[BPX1MP \(mvspause\) 1263](#)  
[BPX1MPC \(mvspocclp\) 1264](#)  
[BPX1MPI \(mvspauseinit\) 1265](#)  
[BPX1MPR \(mprotect\) 1265](#)  
[BPX1MSD \(mvsunsigsetup\) 1265](#)  
[BPX1MSS \(mvssigsetup\) 1266](#)  
[BPX1MSY \(msync\) 1266](#)  
[BPX1MUN \(munmap\) 1266](#)  
[BPX1NIC \(nice\) 1266](#)  
[BPX1OPD \(opendir\) 1267](#)  
[BPX1OPN \(open\) 1267](#)  
[BPX1OPT \(getsockopt or setsockopt\) 1268](#)  
[BPX1PAF \(\\_\\_pid\\_affinity\) 1268](#)  
[BPX1PAS \(pause\) 1269](#)  
[BPX1PCF \(pathconf\) 1269](#)  
[BPX1PCT \(pfsctl\) 1269](#)  
[BPX1PI2\(pipe2\) 1270](#)  
[BPX1PIP \(pipe\) 1270](#)  
[BPX1PLC \(prctl\) 1271](#)  
[BPX1POE \(\\_\\_poe\) 1270](#)  
[BPX1POL \(poll\) 1270](#)  
[BPX1PRL \(prlimit 1271, 1363, 1366\)](#)  
[BPX1PSI \(pthread\\_setinrt\) 1272](#)  
[BPX1PST \(pthread\\_setinrttype\) 1272](#)  
[BPX1PTB \(pthread\\_cancel\) 1272](#)  
[BPX1PTC \(pthread\\_create\) 1272](#)  
[BPX1PTD \(pthread\\_detach\) 1273](#)  
[BPX1PTI \(pthread\\_testinrt\) 1273](#)  
[BPX1PTJ \(pthread\\_join\) 1273](#)  
[BPX1PTK \(pthread\\_kill\) 1274](#)  
[BPX1PTQ \(pthread\\_quiesce\) 1274](#)  
[BPX1PTR \(ptrace\) 1274](#)  
[BPX1PTS \(pthread\\_self\) 1274](#)  
[BPX1PTT \(pthread\\_tag\\_np\) 1275](#)  
[BPX1PTX \(pthread\\_exit\\_and\\_get\) 1275](#)  
[BPX1PVR \(pivot\\_root\) 1275](#)  
[BPX1PWD \(\\_\\_passwd\) 1276](#)  
[BPX1QCT \(msgctl\) 1276](#)  
[BPX1QDB \(querydub\) 1276](#)  
[BPX1QGT \(msgget\) 1276](#)  
[BPX1QRC \(msgrcv\) 1277](#)  
[BPX1QSE \(quiesce\) 1277](#)  
[BPX1QSN \(msgsnd\) 1277](#)  
[BPX1RCV \(recv\) 1278](#)  
[BPX1RD2 \(readdir2\) 1279](#)  
[BPX1RDD \(readdir\) 1278](#)  
[BPX1RDL \(readlink\) 1278](#)  
[BPX1RDV \(readv\) 1279](#)  
[BPX1RDX \(read\\_extlink\) 1279](#)  
[BPX1RED \(read\) 1280](#)  
[BPX1REN \(rename\) 1280](#)  
[BPX1RFM \(recvfrom\) 1281](#)  
[BPX1RLA \(readlinkat\) 1281](#)  
[BPX1RMD \(rmdir\) 1281](#)  
[BPX1RMG \(resource\) 1281](#)  
[BPX1RNA \(renameat\) 1282](#)  
[BPX1RPH \(realpath\) 1283](#)  
[BPX1RW \(Pwrite\) 1283](#)  
[BPX1RWD \(rewinddir\) 1283](#)  
[BPX1SA2 \(\\_\\_sigactionset\) 1284](#)  
[BPX1SCT \(semctl\) 1284](#)

examples of callable services (*continued*)

[BPX1SDD \(setdubdefault\) 1284](#)  
[BPX1SEC \(\\_\\_login\) 1285](#)  
[BPX1SEG \(setegid\) 1285](#)  
[BPX1SEL \(select\) 1285](#)  
[BPX1SEU \(seteuid\) 1286](#)  
[BPX1SF \(send\\_file\) 1286](#)  
[BPX1SGE \(setgrent\) 1286](#)  
[BPX1SGI \(setgid\) 1287](#)  
[BPX1SGQ \(sigqueue\) 1287](#)  
[BPX1SGR \(setgroups\) 1287](#)  
[BPX1SGT \(semget\) 1287](#)  
[BPX1SHT \(shutdown\) 1288](#)  
[BPX1SIA \(sigaction\) 1288](#)  
[BPX1SIN \(server\\_init\) 1288](#)  
[BPX1SIP \(sigpending\) 1289](#)  
[BPX1SLK \(shmlock\) 1289](#)  
[BPX1SLP \(sleep\) 1289](#)  
[BPX1SMF \(smf\\_record\) 1290](#)  
[BPX1SND \(send\) 1291](#)  
[BPX1SNS \(setns\) 1291](#)  
[BPX1SOC \(socket or socketpair\) 1291](#)  
[BPX1SOP \(semop\) 1292](#)  
[BPX1SPB \(queue\\_interrupt\) 1292](#)  
[BPX1SPE \(setpwent\) 1292](#)  
[BPX1SPG \(setpgid\) 1293](#)  
[BPX1SPM \(sigprocmask\) 1293](#)  
[BPX1SPN \(spawn\) 1293](#)  
[BPX1SPR \(setpeer\) 1294](#)  
[BPX1SPW \(server\\_pwu\) 1294](#)  
[BPX1SPY \(setpriority\) 1295](#)  
[BPX1SRG \(setregid\) 1295](#)  
[BPX1SRL \(setrlimit\) 1295](#)  
[BPX1SRU \(setreuid\) 1296](#)  
[BPX1SRX \(srx\\_np\) 1296](#)  
[BPX1SSI \(setsid\) 1296](#)  
[BPX1SSU \(sigsuspend\) 1297](#)  
[BPX1STA \(stat\) 1297](#)  
[BPX1STE \(set\\_timer\\_event\) 1297](#)  
[BPX1STF \(w\\_statvfs\) 1297](#)  
[BPX1STL \(set\\_thread\\_limits\) 1298](#)  
[BPX1STO \(sendto\) 1298](#)  
[BPX1STR \(setitimer\) 1298](#)  
[BPX1STV \(statvfs\) 1299](#)  
[BPX1STW \(sigtimedwait\) 1299](#)  
[BPX1SUI \(setuid\) 1299](#)  
[BPX1SWT \(sigwait\) 1300](#)  
[BPX1SYA \(symlinkat\) 1300](#)  
[BPX1SYC \(sysconf\) 1300](#)  
[BPX1SYF \(syncfs\) 1300](#)  
[BPX1SYM \(symlink\) 1301](#)  
[BPX1SYN \(sync\) 1301](#)  
[BPX1TAF \(MVSThreadAffinity\) 1301](#)  
[BPX1TAK \(takesocket\) 1301](#)  
[BPX1TDR \(tcdrain\) 1302](#)  
[BPX1TFH \(tcflush\) 1302](#)  
[BPX1TFW \(tcflow\) 1302](#)  
[BPX1TGA \(tcgetattr\) 1302](#)  
[BPX1TGC \(tcgetcp\) 1303](#)  
[BPX1TGP \(tcgetgrp\) 1303](#)  
[BPX1TGS \(tcgetsid\) 1303](#)  
[BPX1TIM \(times\) 1303](#)  
[BPX1TLS \(pthread\\_security\\_np\) 1304](#)  
[BPX1TRU \(truncate\) 1304](#)

examples of callable services (*continued*)

[BPX1TSA \(tcsetattr\) 1304](#)  
[BPX1TSB \(tcsendbreak\) 1305](#)  
[BPX1TSC \(tcsetcp\) 1305](#)  
[BPX1TSP \(tcsetpgrp\) 1305](#)  
[BPX1TST \(tcsettables\) 1305](#)  
[BPX1TYN \(ttyname\) 1306](#)  
[BPX1ULA \(unlink\) 1306](#)  
[BPX1ULA \(unlinkat\) 1399](#)  
[BPX1UMK \(umask\) 1307](#)  
[BPX1UMT \(umount\) 1307](#)  
[BPX1UNA \(uname\) 1307](#)  
[BPX1UNL \(unlink\) 1308](#)  
[BPX1UNS \(unshare\) 1308](#)  
[BPX1UPT \(unlockpt\) 1308](#)  
[BPX1UQS \(unquiesce\) 1308](#)  
[BPX1UTI \(utime\) 1309](#)  
[BPX1WAT \(wait\) 1309](#)  
[BPX1WLM \(\\_\\_WLM\) 1309](#)  
[BPX1WRT \(write\) 1310](#)  
[BPX1WRV \(writev\) 1310](#)  
[BPX1WTE \(wait extension\) 1310](#)  
[BPX2ITY \(isatty\) 1254](#)  
[BPX2MNT \(\\_\\_mount\) 1263](#)  
[BPX2OPT \(openstat\) 1267](#)  
[BPX2RMS \(recvmsg\) 1282](#)  
[BPX2SMS \(sendmsg\) 1290](#)  
[BPX2TYN \(ttyname\) 1306](#)  
[BPX4AC4 \(accept4\) 1314, 1315](#)  
[BPX4ACC \(access\) 1313](#)  
[BPX4ACK \(auth\\_check\\_resource\\_np\) 1314](#)  
[BPX4ACP \(accept\) 1315](#)  
[BPX4AIO \(asyncio\) 1315](#)  
[BPX4ALR \(alarm\) 1315](#)  
[BPX4ANR \(accept\\_and\\_recv\) 1316](#)  
[BPX4ASP \(aio\\_suspend\) 1316](#)  
[BPX4ATM \(attach\\_execmvs\) 1316](#)  
[BPX4ATX \(attach\\_exec\) 1317](#)  
[BPX4AUT \(\\_\\_authenticate\) 1317](#)  
[BPX4BAS \(bind with source address selection\) 1318](#)  
[BPX4BND \(bind\) 1318](#)  
[BPX4CCA \(cond\\_cancel\) 1318](#)  
[BPX4CCS \(\\_\\_console\(\)\) 1319](#)  
[BPX4CHA \(chaudit\) 1319](#)  
[BPX4CHD \(chdir\) 1319](#)  
[BPX4CHM \(chmod\) 1319](#)  
[BPX4CHO \(chown\) 1320](#)  
[BPX4CHOA \(fchownat\) 1320](#)  
[BPX4CHP \(chpriority\) 1320](#)  
[BPX4CHR \(chattr\) 1321](#)  
[BPX4CLD \(closedir\) 1322](#)  
[BPX4CLN \(clone\) 1322](#)  
[BPX4CLO \(close\) 1323](#)  
[BPX4CON \(connect\) 1323](#)  
[BPX4CPO \(cond\\_post\) 1323](#)  
[BPX4CRT \(chroot\) 1324](#)  
[BPX4CSE \(cond\\_setup\) 1324](#)  
[BPX4CTW \(cond\\_timed\\_wait\) 1325](#)  
[BPX4CWA \(cond\\_wait\) 1325](#)  
[BPX4DEL \(deleteHFS\) 1325](#)  
[BPX4EFD \(eventfd\) 1326](#)  
[BPX4ENV \(oe\\_env\\_np\) 1326](#)  
[BPX4EPC \(epoll\\_create\) 1326](#)  
[BPX4EPL \(epoll\\_ctl\) 1327](#)

examples of callable services (*continued*)

[BPX4EPW \(epoll\\_wait\) 1327](#)  
[BPX4EXC \(exec\) 1327](#)  
[BPX4EXI \(\\_exit\) 1328](#)  
[BPX4EXM \(execmvs\) 1328](#)  
[BPX4EXT \(extlink\\_np\) 1329](#)  
[BPX4FAI \(freeaddrinfo\) 1329](#)  
[BPX4FCA \(fchaudit\) 1330](#)  
[BPX4FCD \(fchdir\) 1330](#)  
[BPX4FCM \(fchmod\) 1330](#)  
[BPX4FCO \(fchown\) 1330](#)  
[BPX4FCR \(fchattr\) 1331](#)  
[BPX4FCT \(fcntl\) 1331](#)  
[BPX4FLK \(flock\) example 1332](#)  
[BPX4FPC \(fpathconf\) 1332](#)  
[BPX4FRK \(fork\) 1332](#)  
[BPX4FSA \(fchmodat\) 1240](#)  
[BPX4FST \(fstat\) 1333](#)  
[BPX4FSY \(fsync\) 1333](#)  
[BPX4FTR \(ftruncate\) 1334](#)  
[BPX4FTV \(fstatvfs\) 1334](#)  
[BPX4GAI \(getaddrinfo\) 1334](#)  
[BPX4GCL \(getclientid\) 1335](#)  
[BPX4GCW \(getcwd\) 1335](#)  
[BPX4GEG \(getegid\) 1335](#)  
[BPX4GEP \(getpgid\) 1335](#)  
[BPX4GES \(getsid\) 1336](#)  
[BPX4GET \(w\\_getipc\) 1336](#)  
[BPX4GEU \(geteuid\) 1336](#)  
[BPX4GGE \(getgrent\) 1336](#)  
[BPX4GGI \(getgrgid\) 1337](#)  
[BPX4GGN \(getgrnam\) 1337](#)  
[BPX4GGR \(getgroups\) 1338](#)  
[BPX4GHA \(gethostbyaddr\) 1338](#)  
[BPX4GHN \(gethostbyname\) 1338](#)  
[BPX4GID \(getgid\) 1339](#)  
[BPX4GIV \(givesocket\) 1339](#)  
[BPX4GLG \(getlogin\) 1339](#)  
[BPX4GMN \(w\\_getmntent\) 1340](#)  
[BPX4GNI \(getnameinfo\) 1340](#)  
[BPX4GNM \(getpeername or getsockname\) 1341](#)  
[BPX4GPE \(getpwent\) 1341](#)  
[BPX4GPG \(getpgrp\) 1340](#)  
[BPX4GPI \(getpid\) 1341](#)  
[BPX4GPN \(getpwnam\) 1341](#)  
[BPX4GPP \(getppid\) 1342](#)  
[BPX4GPT \(\) 1342](#)  
[BPX4GPU \(getpwuid\) 1342](#)  
[BPX4GPY \(getpriority\) 1342](#)  
[BPX4GRD \(getrandom\) 1343](#)  
[BPX4GRL \(getrlimit\) 1343](#)  
[BPX4GRU \(getrusage\) 1343](#)  
[BPX4GTH \(\\_\\_getthent\) 1344](#)  
[BPX4GTR \(getitimer\) 1344](#)  
[BPX4GUG \(getgroupsbyname\) 1344](#)  
[BPX4GUI \(getuid\) 1345](#)  
[BPX4GWD \(getwd\) 1345](#)  
[BPX4HST \(gethostid or gethostname\) 1345](#)  
[BPX4IAW \(inotify\\_add\\_watch\) 1346](#)  
[BPX4IOC \(w\\_ioctl\) 1347](#)  
[BPX4IPT \(mvsiptaffinity\) 1347](#)  
[BPX4IRM \(inotify\\_rm\\_watch\) 1346](#)  
[BPX4ITY \(isatty\) 1347](#)  
[BPX4KIL \(kill\) 1347](#)



examples of callable services (*continued*)

[BPX4LCO \(lchown\) 1348](#)  
[BPX4LCR \(lchattr\) 1348](#)  
[BPX4LDX \(loadhfs extended\) 1348](#)  
[BPX4LNA \(linkat\) 1350](#)  
[BPX4LNK \(link\) 1351](#)  
[BPX4LOD \(loadhfs\) 1350](#)  
[BPX4LSK \(lseek\) 1351](#)  
[BPX4LSN \(listen\) 1351](#)  
[BPX4LST \(lstat\) 1351](#)  
[BPX4MAT \(shmat\) 1352](#)  
[BPX4MCT \(shmctl\) 1352](#)  
[BPX4MDA \(mkdirat\) 1352](#)  
[BPX4MDT \(shmdt\) 1353](#)  
[BPX4MGT \(shmget\) 1353](#)  
[BPX4MKD \(mkdir\) 1353](#)  
[BPX4MKN \(mknod\) 1354](#)  
[BPX4MMI \(\\_\\_map\\_init\) 1354](#)  
[BPX4MMP \(mmap\) 1355](#)  
[BPX4MMS \(\\_\\_map\\_service\) 1355](#)  
[BPX4MNA \(mknodat\) 1355](#)  
[BPX4MNT \(\\_\\_mount\) 1356](#)  
[BPX4MP \(mvspause\) 1356](#)  
[BPX4MPC \(mvspocclp\) 1356](#)  
[BPX4MPI \(mvspauseinit\) 1357](#)  
[BPX4MPR \(mprotect\) 1357](#)  
[BPX4MSD \(mvunsigsetup\) 1357](#)  
[BPX4MSS \(mvssigsetup\) 1358](#)  
[BPX4MSY \(msync\) 1358](#)  
[BPX4MUN \(munmap\) 1358](#)  
[BPX4NIC \(nice\) 1358](#)  
[BPX4OPD \(opendir\) 1359](#)  
[BPX4OPN \(open\) 1359](#)  
[BPX4OPS \(openstat\) 1359](#)  
[BPX4OPT \(getsockopt or setsockopt\) 1360](#)  
[BPX4PAF \(\\_\\_pid\\_affinity\) 1360](#)  
[BPX4PAS \(pause\) 1361](#)  
[BPX4PCF \(pathconf\) 1361](#)  
[BPX4PCT \(pfsctl\) 1361](#)  
[BPX4PI2\(pipe2\) 1362](#)  
[BPX4PIP \(pipe\) 1362](#)  
[BPX4PLC \(prctl\) 1363](#)  
[BPX4POE \(\\_\\_poe\) 1362](#)  
[BPX4POL \(poll\) 1362](#)  
[BPX4PSI \(pthread\\_setinrt\) 1364](#)  
[BPX4PST \(pthread\\_setinrttype\) 1364](#)  
[BPX4PTB \(pthread\\_cancel\) 1364](#)  
[BPX4PTC \(pthread\\_create\) 1364](#)  
[BPX4PTD \(pthread\\_detach\) 1365](#)  
[BPX4PTI \(pthread\\_testinrt\) 1365](#)  
[BPX4PTJ \(pthread\\_join\) 1365](#)  
[BPX4PTK \(pthread\\_kill\) 1366](#)  
[BPX4PTQ \(pthread\\_quiesce\) 1366](#)  
[BPX4PTR \(ptrace\) 1366](#)  
[BPX4PTS \(pthread\\_self\) 1367](#)  
[BPX4PTT \(pthread\\_tag\\_np\) 1367](#)  
[BPX4PTX \(pthread\\_exit\\_and\\_get\) 1367](#)  
[BPX4PVR \(pivot\\_root\) 1368](#)  
[BPX4PWD \(\\_\\_passwd\) 1368](#)  
[BPX4QCT \(msgctl\) 1368](#)  
[BPX4QDB \(querydub\) 1369](#)  
[BPX4QGT \(msgget\) 1369](#)  
[BPX4QRC \(msgrcv\) 1369](#)  
[BPX4QSE \(quiesce\) 1370](#)

examples of callable services (*continued*)

[BPX4QSN \(msgsnd\) 1370](#)  
[BPX4RCV \(recv\) 1370](#)  
[BPX4RD2 \(readdir2\) 1372](#)  
[BPX4RDD \(readdir\) 1371](#)  
[BPX4RDL \(readlink\) 1371](#)  
[BPX4RDV \(readv\) 1371](#)  
[BPX4RDX \(read\\_extlink\) 1372](#)  
[BPX4RED \(read\) 1372](#)  
[BPX4REN \(rename\) 1373](#)  
[BPX4RFM \(recvfrom\) 1373](#)  
[BPX4RLA \(readlinkat\) 1373](#)  
[BPX4RMD \(rmdir\) 1374](#)  
[BPX4RMG \(resource\) 1374](#)  
[BPX4RMS \(recvmsg\) 1374](#)  
[BPX4RNA \(renameat\) 1375](#)  
[BPX4RPH \(realpath\) 1375](#)  
[BPX4RW \(Pwrite\) 1375](#)  
[BPX4RWD \(rewinddir\) 1376](#)  
[BPX4SA2 \(\\_\\_sigactionset\) 1376](#)  
[BPX4SCT \(semctl\) 1376](#)  
[BPX4SDD \(setdubdefault\) 1377](#)  
[BPX4SEC \(\\_\\_login\) 1377](#)  
[BPX4SEG \(setegid\) 1378](#)  
[BPX4SEL \(select\) 1378](#)  
[BPX4SEU \(seteuid\) 1378](#)  
[BPX4SF \(send\\_file\) 1379](#)  
[BPX4SGE \(setgrent\) 1379](#)  
[BPX4SGI \(setgid\) 1379](#)  
[BPX4SGQ \(sigqueue\) 1379](#)  
[BPX4SGR \(setgroups\) 1380](#)  
[BPX4SGT \(semget\) 1380](#)  
[BPX4SHT \(shutdown\) 1380](#)  
[BPX4SIA \(sigaction\) 1381](#)  
[BPX4SIN \(server\\_init\) 1381](#)  
[BPX4SIP \(sigpending\) 1381](#)  
[BPX4SLK \(shmlock\) 1382](#)  
[BPX4SLP \(sleep\) 1382](#)  
[BPX4SMF \(smf\\_record\) 1382](#)  
[BPX4SMS \(sendmsg\) 1383](#)  
[BPX4SND \(send\) 1383](#)  
[BPX4SNS \(setns\) 1384](#)  
[BPX4SOC \(socket or socketpair\) 1384](#)  
[BPX4SOP \(semop\) 1384](#)  
[BPX4SPB \(queue\\_interrupt\) 1385](#)  
[BPX4SPE \(setpwent\) 1385](#)  
[BPX4SPG \(setpgid\) 1385](#)  
[BPX4SPM \(sigprocmask\) 1385](#)  
[BPX4SPN \(spawn\) 1386](#)  
[BPX4SPR \(setpeer\) 1387](#)  
[BPX4SPW \(server\\_pwu\) 1387](#)  
[BPX4SPY \(setpriority\) 1387](#)  
[BPX4SRG \(setregid\) 1388](#)  
[BPX4SRL \(setrlimit\) 1388](#)  
[BPX4SRU \(setreuid\) 1388](#)  
[BPX4SRX \(srx\\_np\) 1389](#)  
[BPX4SSI \(setsid\) 1389](#)  
[BPX4SSU \(sigsuspend\) 1389](#)  
[BPX4STA \(stat\) 1389](#)  
[BPX4STE \(set\\_timer\\_event\) 1390](#)  
[BPX4STF \(w\\_statvfs\) 1390](#)  
[BPX4STL \(set\\_thread\\_limits\) 1390](#)  
[BPX4STO \(sendto\) 1391](#)  
[BPX4STR \(setitimer\) 1391](#)

examples of callable services (*continued*)

[BPX4STV \(statvfs\) 1391](#)  
[BPX4STW \(sigtimedwait\) 1392](#)  
[BPX4SUI \(setuid\) 1392](#)  
[BPX4SWT \(sigwait\) 1392](#)  
[BPX4SYA \(symlinkat\) 1392](#)  
[BPX4SYC \(sysconf\) 1393](#)  
[BPX4SYF \(syncfs\) 1393](#)  
[BPX4SYM \(symlink\) 1393](#)  
[BPX4SYN \(sync\) 1394](#)  
[BPX4TAF \(MVSThreadAffinity\) 1394](#)  
[BPX4TAK \(takesocket\) 1394](#)  
[BPX4TDR \(tcdrain\) 1394](#)  
[BPX4TFH \(tcflush\) 1395](#)  
[BPX4TFW \(tcflow\) 1395](#)  
[BPX4TGA \(tcgetattr\) 1395](#)  
[BPX4TGC \(tcgetcp\) 1395](#)  
[BPX4TGP \(tcgetpgrp\) 1396](#)  
[BPX4TGS \(tcgetsid\) 1396](#)  
[BPX4TIM \(times\) 1396](#)  
[BPX4TLS \(pthread\\_security\\_np\) 1396](#)  
[BPX4TRU \(truncate\) 1397](#)  
[BPX4TSA \(tcsetattr\) 1397](#)  
[BPX4TSB \(tcsendbreak\) 1397](#)  
[BPX4TSC \(tcsetcp\) 1397](#)  
[BPX4TSP \(tcsetpgrp\) 1398](#)  
[BPX4TST \(tcsettables\) 1398](#)  
[BPX4TYN \(ttyname\) 1399](#)  
[BPX4UMK \(umask\) 1399](#)  
[BPX4UMT \(umount\) 1399](#)  
[BPX4UNA \(uname\) 1400](#)  
[BPX4UNL \(unlink\) 1400](#)  
[BPX4UNS \(unshare\) 1401](#)  
[BPX4UPT \(unlockpt\) 1400](#)  
[BPX4UQS \(unquiesce\) 1400](#)  
[BPX4UTI \(utime\) 1401](#)  
[BPX4WAT \(wait\) 1401](#)  
[BPX4WLM \(\\_\\_WLM\) 1402](#)  
[BPX4WRT \(write\) 1402](#)  
[BPX4WRV \(writev\) 1402](#)  
[BPX4WTE \(wait extension\) 1403](#)  
[chattr \(BPX1CHR\) 1228](#)  
[chattr \(BPX4CHR\) 1321](#)  
[chattrat\(BPX1CRA\) 1230](#)  
[chattrat\(BPX4CRA\) 1323](#)  
[chaudit \(BPX1CHA\) 1225](#)  
[chaudit \(BPX4CHA\) 1319](#)  
[chdir \(BPX1CHD\) 1226](#)  
[chdir \(BPX4CHD\) 1319](#)  
[chmod \(BPX1CHM\) 1226](#)  
[chmod \(BPX4CHM\) 1319](#)  
[chown \(BPX1CHO\) 1226](#)  
[chown \(BPX4CHO\) 1320](#)  
[chpriority \(BPX1CHP\) 1227](#)  
[chpriority \(BPX4CHP\) 1320](#)  
[chroot \(BPX1CRT\) 1230, 1231](#)  
[chroot \(BPX4CRT\) 1324](#)  
[clone\(BPX1CLN\) 1228](#)  
[clone\(BPX4CLN\) 1322](#)  
[close \(BPX1CLO\) 1229](#)  
[close \(BPX4CLO\) 1323](#)  
[closedir \(BPX1CLD\) 1229](#)  
[closedir \(BPX4CLD\) 1322](#)  
[cond\\_cancel \(BPX1CCA\) 1225](#)

examples of callable services (*continued*)

[cond\\_cancel \(BPX4CCA\) 1318](#)  
[cond\\_post \(BPX1CPO\) 1230](#)  
[cond\\_post \(BPX4CPO\) 1323](#)  
[cond\\_setup \(BPX1CSE\) 1231](#)  
[cond\\_setup \(BPX4CSE\) 1324](#)  
[cond\\_timed\\_wait \(BPX1CTW\) 1231](#)  
[cond\\_timed\\_wait \(BPX4CTW\) 1325](#)  
[cond\\_wait \(BPX1CWA\) 1232](#)  
[cond\\_wait \(BPX4CWA\) 1325](#)  
[connect \(BPX1CON\) 1229](#)  
[connect \(BPX4CON\) 1323](#)  
[deleteHFS \(BPX1DEL\) 1232](#)  
[deleteHFS \(BPX4DEL\) 1325](#)  
[epoll\\_create \(BPX1EPC\) 1233](#)  
[epoll\\_create \(BPX4EPC\) 1326](#)  
[epoll\\_create \(BPX4EPW\) 1327](#)  
[epoll\\_ctl \(BPX4EPL\) 1327](#)  
[epoll\\_epl \(BPX4EPL\) 1233](#)  
[epoll\\_wait \(BPX1EPW\) 1234](#)  
[eventfd \(BPX1EFD\) 1232](#)  
[eventfd \(BPX4EFD\) 1326](#)  
[exec \(BPX1EXC\) 1234](#)  
[exec \(BPX1IPT\) 1253](#)  
[exec \(BPX1TAF\) 1301](#)  
[exec \(BPX4EXC\) 1327](#)  
[exec \(BPX4IPT\) 1347](#)  
[exec \(BPX4TAF\) 1394](#)  
[execmvs \(BPX1EXM\) 1235](#)  
[execmvs \(BPX4EXM\) 1328](#)  
[extlink\\_np \(BPX1EXT\) 1235](#)  
[extlink\\_np \(BPX4EXT\) 1329](#)  
[faccessat \(BPX1FAA\) 1235, 1329](#)  
[fchattr \(BPX1FCR\) 1237](#)  
[fchattr \(BPX4FCR\) 1331](#)  
[fchaudit \(BPX1FCA\) 1236](#)  
[fchaudit \(BPX4FCA\) 1330](#)  
[fchdir \(BPX1FCD\) 1237](#)  
[fchdir \(BPX4FCD\) 1330](#)  
[fchmod \(BPX1FCM\) 1237](#)  
[fchmod \(BPX4FCM\) 1330](#)  
[fchmodat \(BPX1CMA\) 1240](#)  
[fchmodat \(BPX1FSA\) 1333](#)  
[fchown \(BPX1FCO\) 1237](#)  
[fchown \(BPX4FCO\) 1330](#)  
[fchownat \(BPX1CHOA\) 1226](#)  
[fchownat \(BPX4CHOA\) 1320](#)  
[fcntl \(BPX1FCT\) 1238](#)  
[fcntl \(BPX4FCT\) 1331](#)  
[flock \(BPX1FLK\) 1239](#)  
[flock \(BPX4FLK\) 1332](#)  
[fork \(BPX1FRK\) 1239](#)  
[fork \(BPX4FRK\) 1332](#)  
[fpathconf \(BPX1FPC\) 1239](#)  
[fpathconf \(BPX4FPC\) 1332](#)  
[freeaddrinfo \(BPX1FAI\) 1236](#)  
[freeaddrinfo \(BPX4FAI\) 1329](#)  
[fstat \(BPX1FST\) 1239](#)  
[fstat \(BPX4FST\) 1333](#)  
[fstatat \(BPX1CMA\) 1227, 1321](#)  
[fstatvfs \(BPX1FTV\) 1240](#)  
[fstatvfs \(BPX4FTV\) 1334](#)  
[fsync \(BPX1FSY\) 1240](#)  
[fsync \(BPX4FSY\) 1333](#)

examples of callable services (*continued*)

[ftruncate \(BPX1FTR\) 1240](#)  
[ftruncate \(BPX4FTR\) 1334](#)  
[getaddrinfo \(BPX1GAI\) 1241](#)  
[getaddrinfo \(BPX4GAI\) 1334](#)  
[getclientid \(BPX1GCL\) 1241](#)  
[getclientid \(BPX4GCL\) 1335](#)  
[getcwd \(BPX1GCW\) 1241](#)  
[getcwd \(BPX4GCW\) 1335](#)  
[getegid \(BPX1GEG\) 1242](#)  
[getegid \(BPX4GEG\) 1335](#)  
[geteuid \(BPX1GEU\) 1243](#)  
[geteuid \(BPX4GEU\) 1336](#)  
[getgid \(BPX1GID\) 1245](#)  
[getgid \(BPX4GID\) 1339](#)  
[getgrent \(BPX1GGE\) 1243](#)  
[getgrent \(BPX4GGE\) 1336](#)  
[getgrgid \(BPX1GGI\) 1243](#)  
[getgrgid \(BPX4GGI\) 1337](#)  
[getgrnam \(BPX1GGN\) 1244](#)  
[getgrnam \(BPX4GGN\) 1337](#)  
[getgroups \(BPX1GGR\) 1244](#)  
[getgroups \(BPX4GGR\) 1338](#)  
[getgroupsbyname \(BPX1GUG\) 1251](#)  
[getgroupsbyname \(BPX4GUG\) 1344](#)  
[gethostbyaddr \(BPX1GHA\) 1244](#)  
[gethostbyaddr \(BPX4GHA\) 1338](#)  
[gethostbyname \(BPX1GHN\) 1245](#)  
[gethostbyname \(BPX4GHN\) 1338](#)  
[gethostid or gethostname \(BPX1HST\) 1252](#)  
[gethostid or gethostname \(BPX4HST\) 1345](#)  
[getitimer \(BPX1GTR\) 1251](#)  
[getitimer \(BPX4GTR\) 1344](#)  
[getlogin \(BPX1GLG\) 1246](#)  
[getlogin \(BPX4GLG\) 1339](#)  
[getnameinfo \(BPX1GNI\) 1246](#)  
[getnameinfo \(BPX4GNI\) 1340](#)  
[getpeername or getsockname \(BPX1GNM\) 1247](#)  
[getpeername or getsockname \(BPX4GNM\) 1341](#)  
[getpgid \(BPX1GEP\) 1242](#)  
[getpgid \(BPX4GEP\) 1335](#)  
[getpgrp \(BPX1GPG\) 1247](#)  
[getpgrp \(BPX4GPG\) 1340](#)  
[getpid \(BPX1GPI\) 1248](#)  
[getpid \(BPX4GPI\) 1341](#)  
[getppid \(BPX1GPP\) 1248](#)  
[getppid \(BPX4GPP\) 1342](#)  
[getpriority \(BPX1GPY\) 1249](#)  
[getpriority \(BPX4GPY\) 1342](#)  
[getpwent \(BPX1GPE\) 1247](#)  
[getpwent \(BPX4GPE\) 1341](#)  
[getpwnam \(BPX1GPN\) 1248](#)  
[getpwnam \(BPX4GPN\) 1341](#)  
[getpwuid \(BPX1GPU\) 1249](#)  
[getpwuid \(BPX4GPU\) 1342](#)  
[getrandom \(BPX1GRD\) 1250](#)  
[getrandom \(BPX4GRD\) 1343](#)  
[getrlimit \(BPX1GRL\) 1250](#)  
[getrlimit \(BPX4GRL\) 1343](#)  
[getrusage \(BPX1GRU\) 1250](#)  
[getrusage \(BPX4GRU\) 1343](#)  
[getsid \(BPX1GES\) 1242](#)  
[getsid \(BPX4GES\) 1336](#)  
[getsockopt or setsockopt \(BPX1OPT\) 1268](#)

examples of callable services (*continued*)

[getsockopt or setsockopt \(BPX4OPT\) 1360](#)  
[getuid \(BPX1GUI\) 1252](#)  
[getuid \(BPX4GUI\) 1345](#)  
[getwd \(BPX1GWD\) 1252](#)  
[getwd \(BPX4GWD\) 1345](#)  
[givesocket \(BPX1GIV\) 1245](#)  
[givesocket \(BPX4GIV\) 1339](#)  
[grantpt \(BPX1GPT\) 1249](#)  
[grantpt \(BPX4GPT\) 1342](#)  
[inotify\\_add\\_watch \(BPX1IAW\) 1252](#)  
[inotify\\_add\\_watch \(BPX4IAW\) 1346](#)  
[inotify\\_init \(BPX1INI\) 1253, 1346](#)  
[inotify\\_rm\\_watch \(BPX1IRM\) 1254](#)  
[inotify\\_rm\\_watch \(BPX4IRM\) 1346](#)  
[isatty \(BPX1ITY\) 1254](#)  
[isatty \(BPX2ITY\) 1254](#)  
[isatty \(BPX4ITY\) 1347](#)  
[kill \(BPX1KIL\) 1254](#)  
[kill \(BPX4KIL\) 1347](#)  
[lchattr \(BPX1LCR\) 1255](#)  
[lchattr \(BPX4LCR\) 1348](#)  
[lchown \(BPX1LCO\) 1255](#)  
[lchown \(BPX4LCO\) 1348](#)  
[link \(BPX1LNK\) 1258](#)  
[link \(BPX4LNK\) 1351](#)  
[linkat \(BPX1LNA\) 1257](#)  
[linkat \(BPX4LNA\) 1350](#)  
[loadhfs \(BPX1LOD\) 1257](#)  
[loadhfs \(BPX4LOD\) 1350](#)  
[loadhfs extended \(BPX1LDX\) 1255](#)  
[loadhfs extended \(BPX4LDX\) 1348](#)  
[lseek \(BPX1LSK\) 1258](#)  
[lseek \(BPX4LSK\) 1351](#)  
[lstat \(BPX1LST\) 1258](#)  
[lstat \(BPX4LST\) 1351](#)  
[mkdir \(BPX1MKD\) 1260](#)  
[mkdir \(BPX4MKD\) 1353](#)  
[mkdirat \(BPX1MDA\) 1259](#)  
[mkdirat \(BPX4MDA\) 1352](#)  
[mknod \(BPX1MKN\) 1261](#)  
[mknod \(BPX4MKN\) 1354](#)  
[mknodat \(BPX1MNA\) 1262](#)  
[mknodat \(BPX4MNA\) 1355](#)  
[mmap \(BPX1MMP\) 1262](#)  
[mmap \(BPX4MMP\) 1355](#)  
[mount \(BPX1MNT\) 1263](#)  
[mprotect \(BPX1MPR\) 1265](#)  
[mprotect \(BPX4MPR\) 1357](#)  
[msgctl \(BPX1QCT\) 1276](#)  
[msgctl \(BPX4QCT\) 1368](#)  
[msgget \(BPX1QGT\) 1276](#)  
[msgget \(BPX4QGT\) 1369](#)  
[msgrcv \(BPX1QRC\) 1277](#)  
[msgrcv \(BPX4QRC\) 1369](#)  
[msgsnd \(BPX1QSN\) 1277](#)  
[msgsnd \(BPX4QSN\) 1370](#)  
[msync \(BPX1MSY\) 1266](#)  
[msync \(BPX4MSY\) 1358](#)  
[munmap \(BPX1MUN\) 1266](#)  
[munmap \(BPX4MUN\) 1358](#)  
[mvspause \(BPX1MP\) 1263](#)  
[mvspause \(BPX4MP\) 1356](#)  
[mvspauseinit \(BPX1MPI\) 1265](#)



examples of callable services (*continued*)

[mvspauseinit \(BPX4MPI\) 1357](#)  
[mvspocclp \(BPX1MPC\) 1264](#)  
[mvspocclp \(BPX4MPC\) 1356](#)  
[mvssigsetup \(BPX1MSS\) 1266](#)  
[mvssigsetup \(BPX4MSS\) 1358](#)  
[mvunsigsetup \(BPX1MSD\) 1265](#)  
[mvunsigsetup \(BPX4MSD\) 1357](#)  
[nice \(BPX1NIC\) 1266](#)  
[nice \(BPX4NIC\) 1358](#)  
[oe\\_env\\_np \(BPX1ENV\) 1233](#)  
[oe\\_env\\_np \(BPX4ENV\) 1326](#)  
[open \(BPX1OPN\) 1267](#)  
[open \(BPX4OPN\) 1359](#)  
[opendir \(BPX1OPD\) 1267](#)  
[opendir \(BPX4OPD\) 1359](#)  
[openstat \(BPX2OPT\) 1267](#)  
[openstat \(BPX4OPS\) 1359](#)  
[pathconf \(BPX1PCF\) 1269](#)  
[pathconf \(BPX4PCF\) 1361](#)  
[pause \(BPX1PAS\) 1269](#)  
[pause \(BPX4PAS\) 1361](#)  
[pfsctl \(BPX1PCT\) 1269](#)  
[pfsctl \(BPX4PCT\) 1361](#)  
[pipe \(BPX1PIP\) 1270](#)  
[pipe \(BPX1POE\) 1270](#)  
[pipe \(BPX4PIP\) 1362](#)  
[pipe \(BPX4POE\) 1362](#)  
[pipe2 \(BPX1PI2\) 1270](#)  
[pipe2 \(BPX4PI2\) 1362](#)  
[pivot\\_root \(BPX1PVR\) 1275](#)  
[pivot\\_root \(BPX4PVR\) 1368](#)  
[poll \(BPX1POL\) 1270](#)  
[poll \(BPX4POL\) 1362](#)  
[prctl \(BPX1PLC\) 1271](#)  
[prctl \(BPX4PLC\) 1363](#)  
[prlimit \(BPX1PRL\) 1271, 1363, 1366](#)  
[pthread\\_cancel \(BPX1PTB\) 1272](#)  
[pthread\\_cancel \(BPX4PTB\) 1364](#)  
[pthread\\_create \(BPX1PTC\) 1272](#)  
[pthread\\_create \(BPX4PTC\) 1364](#)  
[pthread\\_detach \(BPX1PTD\) 1273](#)  
[pthread\\_detach \(BPX4PTD\) 1365](#)  
[pthread\\_exit\\_and\\_get \(BPX1PTX\) 1275](#)  
[pthread\\_exit\\_and\\_get \(BPX4PTX\) 1367](#)  
[pthread\\_join \(BPX1PTJ\) 1273](#)  
[pthread\\_join \(BPX4PTJ\) 1365](#)  
[pthread\\_kill \(BPX1PTK\) 1274](#)  
[pthread\\_kill \(BPX4PTK\) 1366](#)  
[pthread\\_quiesce \(BPX1PTQ\) 1274](#)  
[pthread\\_quiesce \(BPX4PTQ\) 1366](#)  
[pthread\\_security\\_np \(BPX1TLS\) 1304](#)  
[pthread\\_security\\_np \(BPX4TLS\) 1396](#)  
[pthread\\_self \(BPX1PTS\) 1274](#)  
[pthread\\_self \(BPX4PTS\) 1367](#)  
[pthread\\_setintr \(BPX1PSI\) 1272](#)  
[pthread\\_setintr \(BPX4PSI\) 1364](#)  
[pthread\\_setintrtype \(BPX1PST\) 1272](#)  
[pthread\\_setintrtype \(BPX4PST\) 1364](#)  
[pthread\\_tag\\_np \(BPX1PTT\) 1275](#)  
[pthread\\_tag\\_np \(BPX4PTT\) 1367](#)  
[pthread\\_testintr \(BPX1PTI\) 1273](#)  
[pthread\\_testintr \(BPX4PTI\) 1365](#)  
[ptrace \(BPX1PTR\) 1274](#)

examples of callable services (*continued*)

[ptrace \(BPX4PTR\) 1366](#)  
[Pwrite \(BPX1RW\) 1283](#)  
[Pwrite \(BPX4RW\) 1375](#)  
[querydub \(BPX1QDB\) 1276](#)  
[querydub \(BPX4QDB\) 1369](#)  
[queue\\_interrupt \(BPX1SPB\) 1292](#)  
[queue\\_interrupt \(BPX4SPB\) 1385](#)  
[quiesce \(BPX1QSE\) 1277](#)  
[quiesce \(BPX4QSE\) 1370](#)  
[read \(BPX1RED\) 1280](#)  
[read \(BPX4RED\) 1372](#)  
[read\\_extlink \(BPX1RDX\) 1279](#)  
[read\\_extlink \(BPX4RDX\) 1372](#)  
[readdir \(BPX1RDD\) 1278](#)  
[readdir \(BPX4RDD\) 1371](#)  
[readdir2 \(BPX1RD2\) 1279](#)  
[readdir2 \(BPX4RD2\) 1372](#)  
[readlink \(BPX1RDL\) 1278](#)  
[readlink \(BPX4RDL\) 1371](#)  
[readlinkat \(BPX1RLA\) 1281](#)  
[readlinkat \(BPX4RLA\) 1373](#)  
[readv \(BPX1RDV\) 1279](#)  
[readv \(BPX4RDV\) 1371](#)  
[realpath \(BPX1RPH\) 1283](#)  
[realpath \(BPX4RPH\) 1375](#)  
[recv \(BPX1RCV\) 1278](#)  
[recv \(BPX4RCV\) 1370](#)  
[recvfrom \(BPX1RFM\) 1281](#)  
[recvfrom \(BPX4RFM\) 1373](#)  
[recvmsg \(BPX2RMS\) 1282](#)  
[recvmsg \(BPX4RMS\) 1374](#)  
[reentrant entry 1219, 1313](#)  
[reentrant return linkage 1311, 1403](#)  
[rename \(BPX1REN\) 1280](#)  
[rename \(BPX4REN\) 1373](#)  
[renameat \(BPX1RNA\) 1282](#)  
[renameat \(BPX4RNA\) 1375](#)  
[resource \(BPX1RMG\) 1281](#)  
[resource \(BPX4RMG\) 1374](#)  
[rewinddir \(BPX1RWD\) 1283](#)  
[rewinddir \(BPX4RWD\) 1376](#)  
[rmdir \(BPX1RMD\) 1281](#)  
[rmdir \(BPX4RMD\) 1374](#)  
[select \(BPX1SEL\) 1285](#)  
[select \(BPX4SEL\) 1378](#)  
[semctl \(BPX1SCT\) 1284](#)  
[semctl \(BPX4SCT\) 1376](#)  
[semget \(BPX1SGT\) 1287](#)  
[semget \(BPX4SGT\) 1380](#)  
[semop \(BPX1SOP\) 1292](#)  
[semop \(BPX4SOP\) 1384](#)  
[send \(BPX1SND\) 1291](#)  
[send \(BPX4SND\) 1383](#)  
[send\\_file \(BPX1SF\) 1286](#)  
[send\\_file \(BPX4SF\) 1379](#)  
[sendmsg \(BPX2SMS\) 1290](#)  
[sendmsg \(BPX4SMS\) 1383](#)  
[sendto \(BPX1STO\) 1298](#)  
[sendto \(BPX4STO\) 1391](#)  
[server\\_init \(BPX1SIN\) 1288](#)  
[server\\_init \(BPX4SIN\) 1381](#)  
[server\\_pwu \(BPX1SPW\) 1294](#)  
[server\\_pwu \(BPX4SPW\) 1387](#)

examples of callable services (*continued*)

[set\\_thread\\_limits \(BPX1STL\) 1298](#)  
[set\\_thread\\_limits \(BPX4STL\) 1390](#)  
[setdubdefault \(BPX1SEG\) 1284](#)  
[setdubdefault \(BPX4SEG\) 1377](#)  
[setegid \(BPX1SEG\) 1285](#)  
[setegid \(BPX4SEG\) 1378](#)  
[seteuid \(BPX1SEU\) 1286](#)  
[seteuid \(BPX4SEU\) 1378](#)  
[setgid \(BPX1SGI\) 1287](#)  
[setgid \(BPX4SGI\) 1379](#)  
[setgrent \(BPX1SGE\) 1286](#)  
[setgrent \(BPX4SGE\) 1379](#)  
[setgroups \(BPX1SGR\) 1287](#)  
[setgroups \(BPX4SGR\) 1380](#)  
[setitimer \(BPX1STR\) 1298](#)  
[setitimer \(BPX4STR\) 1391](#)  
[setns \(BPXA4SNS\) 1384](#)  
[setns\(BPX1SNS\) 1291](#)  
[setpeer \(BPX1SPR\) 1294](#)  
[setpeer \(BPX4SPR\) 1387](#)  
[setpgid \(BPX1SPG\) 1293](#)  
[setpgid \(BPX4SPG\) 1385](#)  
[setpriority \(BPX1SPY\) 1295](#)  
[setpriority \(BPX4SPY\) 1387](#)  
[setpwent \(BPX1SPE\) 1292](#)  
[setpwent \(BPX4SPE\) 1385](#)  
[setregid \(BPX1SRG\) 1295](#)  
[setregid \(BPX4SRG\) 1388](#)  
[setreuid \(BPX1SRU\) 1296](#)  
[setreuid \(BPX4SRU\) 1388](#)  
[setrlimit \(BPX1SRL\) 1295](#)  
[setrlimit \(BPX4SRL\) 1388](#)  
[setsid \(BPX1SSI\) 1296](#)  
[setsid \(BPX1STE\) 1297](#)  
[setsid \(BPX4SSI\) 1389](#)  
[setsid \(BPX4STE\) 1390](#)  
[setuid \(BPX1SUI\) 1299](#)  
[setuid \(BPX4SUI\) 1392](#)  
[shmat \(BPX1MAT\) 1259](#)  
[shmat \(BPX4MAT\) 1352](#)  
[shmctl\(BPX1MCT\) 1259](#)  
[shmctl\(BPX4MCT\) 1352](#)  
[shmdt \(BPX1MDT\) 1260](#)  
[shmdt \(BPX4MDT\) 1353](#)  
[shmem\\_lock \(BPX1SLK\) 1289](#)  
[shmem\\_lock \(BPX4SLK\) 1382](#)  
[shmget \(BPX1MGT\) 1260](#)  
[shmget \(BPX4MGT\) 1353](#)  
[shutdown \(BPX1SHT\) 1288](#)  
[shutdown \(BPX4SHT\) 1380](#)  
[sigaction \(BPX1SIA\) 1288](#)  
[sigaction \(BPX4SIA\) 1381](#)  
[sigpending \(BPX1SIP\) 1289](#)  
[sigpending \(BPX4SIP\) 1381](#)  
[sigprocmask \(BPX1SPM\) 1293](#)  
[sigprocmask \(BPX4SPM\) 1385](#)  
[sigqueue \(BPX1SGQ\) 1287](#)  
[sigqueue \(BPX4SGQ\) 1379](#)  
[sigsuspend \(BPX1SSU\) 1297](#)  
[sigsuspend \(BPX4SSU\) 1389](#)  
[sigtimedwait \(BPX1STW\) 1299](#)  
[sigtimedwait \(BPX4STW\) 1392](#)  
[sigwait \(BPX1SWT\) 1300](#)

examples of callable services (*continued*)

[sigwait \(BPX4SWT\) 1392](#)  
[sleep \(BPX1SLP\) 1289](#)  
[sleep \(BPX4SLP\) 1382](#)  
[socket or socketpair \(BPX1SOC\) 1291](#)  
[socket or socketpair \(BPX4SOC\) 1384](#)  
[spawn \(BPX1EXC\) 1293](#)  
[spawn \(BPX4SPN\) 1386](#)  
[srx\\_np \(BPX1SRX\) 1296](#)  
[srx\\_np \(BPX4SRX\) 1389](#)  
[stat \(BPX1STA\) 1297](#)  
[stat \(BPX4STA\) 1389](#)  
[statvfs \(BPX1STV\) 1299](#)  
[statvfs \(BPX4STV\) 1391](#)  
[symlink \(BPX1SYM\) 1301](#)  
[symlink \(BPX4SYM\) 1393](#)  
[symlinkat \(BPX1SYA\) 1300](#)  
[symlinkat \(BPX4SYA\) 1392](#)  
[sync \(BPX1SYN\) 1301](#)  
[sync \(BPX4SYN\) 1394](#)  
[syncfs \(BPX1SYF\) 1300](#)  
[syncfs \(BPX4SYF\) 1393](#)  
[sysconf \(BPX1SYC\) 1300](#)  
[sysconf \(BPX4SYC\) 1393](#)  
[takesocket \(BPX1TAK\) 1301](#)  
[takesocket \(BPX4TAK\) 1394](#)  
[tcdrain \(BPX1TDR\) 1302](#)  
[tcdrain \(BPX4TDR\) 1394](#)  
[tcflow \(BPX1TFW\) 1302](#)  
[tcflow \(BPX4TFW\) 1395](#)  
[tcflush \(BPX1TFH\) 1302](#)  
[tcflush \(BPX4TFH\) 1395](#)  
[tcgetattr \(BPX1TGA\) 1302](#)  
[tcgetattr \(BPX4TGA\) 1395](#)  
[tcgetcp \(BPX1TGC\) 1303](#)  
[tcgetcp \(BPX4TGC\) 1395](#)  
[tcgetpgrp \(BPX1TGP\) 1303](#)  
[tcgetpgrp \(BPX4TGP\) 1396](#)  
[tcgetsid \(BPX1TGS\) 1303](#)  
[tcgetsid \(BPX4TGS\) 1396](#)  
[tcsendbreak \(BPX1TSB\) 1305](#)  
[tcsendbreak \(BPX4TSB\) 1397](#)  
[tcsetattr \(BPX1TSA\) 1304](#)  
[tcsetattr \(BPX4TSA\) 1397](#)  
[tcsetcp \(BPX1TSC\) 1305](#)  
[tcsetcp \(BPX4TSC\) 1397](#)  
[tcsetpgrp \(BPX1TSP\) 1305](#)  
[tcsetpgrp \(BPX4TSP\) 1398](#)  
[tcsettables \(BPX1TST\) 1305](#)  
[tcsettables \(BPX4TST\) 1398](#)  
[times \(BPX1TIM\) 1303](#)  
[times \(BPX4TIM\) 1396](#)  
[truncate \(BPX1TRU\) 1304](#)  
[truncate \(BPX4TRU\) 1397](#)  
[ttyname \(BPX1TYN\) 1306](#)  
[ttyname \(BPX2TYN\) 1306](#)  
[ttyname \(BPX4TYN\) 1399](#)  
[umask \(BPX1UMK\) 1307](#)  
[umask \(BPX4UMK\) 1399](#)  
[umount \(BPX1UMT\) 1307](#)  
[umount \(BPX4UMT\) 1399](#)  
[uname \(BPX1UNA\) 1307](#)  
[uname \(BPX4UNA\) 1400](#)  
[unlink \(BPX1UNL\) 1308](#)

examples of callable services (*continued*)

- unlink (BPX4UNL) [1400](#)
- unlink(BPX1ULA) [1306](#)
- unlinkat(BPX1ULA) [1399](#)
- unlockpt (BPX1UPT) [1308](#)
- unlockpt (BPX4UPT) [1400](#)
- unquiesce (BPX1UQS) [1308](#)
- unquiesce (BPX4UQS) [1400](#)
- unshare (BPX1UNS) [1308](#)
- unshare (BPX4UNS) [1401](#)
- utime (BPX1UTI) [1309](#)
- utime (BPX4UTI) [1401](#)
- w\_getipc (BPX1GET) [1242](#)
- w\_getipc (BPX4GET) [1336](#)
- w\_getmntent (BPX1GMN) [1246](#)
- w\_getmntent (BPX4GMN) [1340](#)
- w\_getpsent (BPX1GPS) [1248](#)
- w\_ioctl (BPX1IOC) [1253](#)
- w\_ioctl (BPX4IOC) [1347](#)
- w\_statvfs (BPX1STF) [1297](#)
- w\_statvfs (BPX4STF) [1390](#)
- wait (BPX1WAT) [1309](#)
- wait (BPX1WTE) [1310](#)
- wait (BPX4WAT) [1401](#)
- wait (BPX4WTE) [1403](#)
- write (BPX1WRT) [1310](#)
- write (BPX4WRT) [1402](#)
- writew (BPX1WRV) [1310](#)
- writew (BPX4WRV) [1402](#)
- Examples of callable services
  - send (BPX1SMF) [1290](#)
  - send (BPX4SMF) [1382](#)
- examples of callable services
  - Listen (BPX1LSN) [1258](#)
  - Listen (BPX4LSN) [1351](#)
- exec (BPX1EXC, BPX4EXC) service [163](#)
- exec (BPX1EXC) service
  - example [1234](#)
- exec (BPX1IPT) service
  - example [1253](#)
- exec (BPX1TAF) service
  - example [1301](#)
- exec (BPX4EXC) service
  - example [1327](#)
- exec (BPX4IPT) service
  - example [1347](#)
- exec (BPX4TAF) service
  - example [1394](#)
- execmvs (BPX1EXM, BPX4EXM) service [175](#)
- execmvs (BPX1EXM) service
  - example [1235](#)
- execmvs (BPX4EXM) service
  - example [1328](#)
- execution
  - MVS program [58](#), [175](#)
  - program [48](#), [107](#), [148](#), [163](#), [287](#), [295](#), [300](#), [313](#), [315](#), [317](#), [393](#), [399](#), [497](#), [768](#), [779](#), [784](#), [786](#), [789](#), [869](#)
  - suspend process [859](#)
- execution on IPT
  - program [476](#)
- exit points for syscalls [1439](#)
- exit routines
  - environment [1440](#)
  - managing [1439](#)

exits

- enabling [1439](#)
- installation [1435](#)
- syscall [1439](#)
- external link
  - read value [663](#)
- external name
  - create symbolic link to [183](#)
- extlink\_np (BPX1EXT, BPX4EXT) service [183](#)
- extlink\_np (BPX1EXT) service
  - example [1235](#)
- extlink\_np (BPX4EXT) service
  - example [1329](#)

**F**

- faccessat
  - check user permission [186](#)
- faccessat (BPX1FAA, BPX4ACC) service [186](#)
- faccessat (BPX1FAA) service
  - example [1235](#), [1329](#)
- fchattr (BPX1FCR, BPX4FCR) service [190](#)
- fchattr (BPX1FCR) service
  - example [1237](#)
- fchattr (BPX4FCR) service
  - example [1331](#)
- fchaudit (BPX1FCA, BPX4FCA) service [199](#)
- fchaudit (BPX1FCA) service
  - example [1236](#)
- fchaudit (BPX4FCA) service
  - example [1330](#)
- fchdir (BPX1FCD, BPX4FCD) service [202](#)
- fchdir (BPX1FCD) service
  - example [1237](#)
- fchdir (BPX4FCD) service
  - example [1330](#)
- fchmod (BPX1FCM, BPX4FCM) service [204](#)
- fchmod (BPX1FCM) service
  - example [1237](#)
- fchmod (BPX4FCM) service
  - example [1330](#)
- fchmodat (BPX1CMA, BPX4CMA) service [206](#)
- fchmodat (BPX1CMA) service
  - example [1227](#), [1240](#), [1321](#)
- fchown (BPX1FCO, BPX4FCO) service [210](#)
- fchown (BPX1FCO) service
  - example [1237](#)
- fchown (BPX4FCO) service
  - example [1330](#)
- fchownat (BPX1COA, BPX4COA) service [213](#)
- fchownat service
  - example [1226](#), [1320](#)
- fcntl (BPX1FCT, BPX4FCT) service [216](#)
- fcntl (BPX1FCT) service
  - example [1238](#)
- fcntl (BPX4FCT) service
  - example [1331](#)
- file
  - change audit flags
    - by descriptor [199](#)
  - change offset [406](#)
  - change the group
    - by descriptor [210](#)
  - change the owner

- file (*continued*)
  - change the owner (*continued*)
    - by descriptor [210](#)
  - changing attributes [91](#)
  - check availability [24](#)
  - check user permission [186](#)
  - close [121](#)
  - create FIFO [427](#)
  - create special character [427](#)
  - determine
    - configurable limit [233](#), [535](#)
    - path name variable [233](#), [535](#)
  - link created [383](#)
  - link created to by using a directory [387](#)
  - obtain status
    - by descriptor [238](#)
  - obtain status information [409](#), [893](#)
  - open and create descriptor [511](#)
  - open and obtain status information [526](#)
  - read [572](#), [652](#)
  - register interest in
    - by descriptor [1003](#)
    - by path name [1025](#)
  - rename [690](#)
  - renaming by using at directory [694](#)
  - represents a terminal [357](#), [359](#)
  - send on a socket [730](#)
  - truncate [249](#), [953](#)
  - write from a buffer to a [1032](#)
  - write to [572](#)
- file descriptor
  - created [1267](#), [1270](#), [1359](#), [1362](#)
- file descriptors
  - control [216](#)
- file mode
  - change
    - by descriptor [204](#)
- file mode creation mask
  - set or return [960](#)
- file system
  - make available [444](#), [447](#), [975](#)
  - mounted
    - information [995](#)
  - obtain status [244](#), [896](#), [1029](#)
  - quiesce [650](#)
  - remove virtual [961](#)
- file tree
  - remove file system from [961](#)
- flags
  - audit
    - change file [95](#)
    - change file by descriptor [199](#)
    - file descriptor [216](#)
    - file status [216](#)
- flock (BPX1FLK, BPX4FLK) service [225](#)
- flock(BPX1FLK) service
  - example [1239](#)
- flock(BPX4FLK) service
  - example [1332](#)
- flush
  - terminal buffer [923](#)
- foreground
  - obtain process group ID [931](#)
  - set process group ID [944](#)

- fork (BPX1FRK, BPX4FRK) service [227](#)
- fork (BPX1FRK) service
  - example [1239](#)
- fork (BPX4FRK) service
  - example [1332](#)
- fpathconf (BPX1FPC, BPX4FPC) service [233](#)
- fpathconf (BPX1FPC) service
  - example [1239](#)
- fpathconf (BPX4FPC) service
  - example [1332](#)
- free
  - Addr\_Info structures [236](#)
- freeaddrinfo (BPX1FAI, BPX4FAI) service [236](#)
- freeaddrinfo (BPX1FAI) service
  - example [1236](#)
- freeaddrinfo (BPX4FAI) service
  - example [1329](#)
- fstat (BPX1FST, BPX4FST) service [238](#)
- fstat (BPX1FST) service
  - example [1239](#)
- fstat (BPX4FST) service
  - example [1333](#)
- fstatat (BPX1FSA, BPX4FSA) service [241](#)
- fstatat (BPX1FSA) service
  - example [1333](#)
- fstatvfs (BPX1FTV, BPX4FTV) service [244](#)
- fstatvfs (BPX1FTV) service
  - example [1240](#)
- fstatvfs (BPX4FTV) service
  - example [1334](#)
- fsync (BPX1FSY, BPX4FSY) service [247](#)
- fsync (BPX1FSY) service
  - example [1240](#)
- fsync (BPX4FSY) service
  - example [1333](#)
- ftruncate (BPX1FTR, BPX4FTR) service [249](#)
- ftruncate (BPX1FTR) service
  - example [1240](#)
- ftruncate (BPX4FTR) service
  - example [1334](#)
- functional recovery routine (FRR) [5](#)

## G

- get
  - terminal code page [928](#)
- getaddrinfo (BPX1GAI, BPX4GAI) service [251](#)
- getaddrinfo (BPX1GAI) service
  - example [1241](#)
- getaddrinfo (BPX4GAI) service
  - example [1334](#)
- getclientid (BPX1GCL, BPX4GCL) service [258](#)
- getclientid (BPX1GCL) service
  - example [1241](#)
- getclientid (BPX4GCL) service
  - example [1335](#)
- getcwd (BPX1GCW, BPX4GCW) service [260](#)
- getcwd (BPX1GCW) service
  - example [1241](#)
- getcwd (BPX4GCW) service
  - example [1335](#)
- getegid (BPX1GEG, BPX4GEG) service [263](#)
- getegid (BPX1GEG) service
  - example [1242](#)

getegid (BPX4GEG) service  
     example [1335](#)  
 geteuid (BPX1GEU, BPX4GEU) service [264](#)  
 geteuid (BPX1GEU) service  
     example [1243](#)  
 geteuid (BPX4GEU) service  
     example [1336](#)  
 getgid (BPX1GID, BPX4GID) service [265](#)  
 getgid (BPX1GID) service  
     example [1245](#)  
 getgid (BPX4GID) service  
     example [1339](#)  
 getgrent (BPX1GGE, BPX4GGE) service [266](#)  
 getgrent (BPX1GGE) service  
     example [1243](#)  
 getgrent (BPX4GGE) service  
     example [1336](#)  
 getgrgid (BPX1GGI, BPX4GGI) service [268](#)  
 getgrgid (BPX1GGI) service  
     example [1243](#)  
 getgrgid (BPX4GGI) service  
     example [1337](#)  
 getgrnam (BPX1GGN, BPX4GGN) service [271](#)  
 getgrnam (BPX1GGN) service  
     example [1244](#)  
 getgrnam (BPX4GGN) service  
     example [1337](#)  
 getgroups (BPX1GGR, BPX4GGR) service [274](#)  
 getgroups (BPX1GGR) service  
     example [1244](#)  
 getgroups (BPX4GGR) service  
     example [1338](#)  
 getgroupsbyname (BPX1GUG, BPX4GUG) service [276](#)  
 getgroupsbyname (BPX1GUG) service  
     example [1251](#)  
 getgroupsbyname (BPX4GUG) service  
     example [1344](#)  
 gethostbyaddr (BPX1GHA, BPX4GHA) service [279](#)  
 gethostbyaddr (BPX1GHA) service  
     example [1244](#)  
 gethostbyaddr (BPX4GHA) service  
     example [1338](#)  
 gethostbyname (BPX1GHN, BPX4GHN) service [282](#)  
 gethostbyname (BPX1GHN) service  
     example [1245](#)  
 gethostbyname (BPX4GHN) service  
     example [1338](#)  
 gethostid or gethostname (BPX1HST, BPX4HST) service [285](#)  
 gethostid or gethostname (BPX1HST) service  
     example [1252](#)  
 gethostid or gethostname (BPX4HST) service  
     example [1345](#)  
 getitimer (BPX1GTR, BPX4GTR) service [287](#)  
 getitimer (BPX1GTR) service  
     example [1251](#)  
 getitimer (BPX4GTR) service  
     example [1344](#)  
 getlogin (BPX1GLG, BPX4GLG) service [289](#)  
 getlogin (BPX1GLG) service  
     example [1246](#)  
 getlogin (BPX4GLG) service  
     example [1339](#)  
 getnameinfo (BPX1GNI, BPX4GNI) service [291](#)  
 getnameinfo (BPX1GNI) service  
     example [1246](#)  
 getnameinfo (BPX4GNI) service  
     example [1340](#)  
 getpeername or getsockname (BPX1GNM) service  
     example [1247](#)  
 getpeername or getsockname (BPX4GNM) service  
     example [1341](#)  
 getpgid (BPX1GEP, BPX4GEP) service [295](#)  
 getpgid (BPX1GEP) service  
     example [1242](#)  
 getpgid (BPX4GEP) service  
     example [1335](#)  
 getpgrp (BPX1GPG, BPX4GPG) service [297](#)  
 getpgrp (BPX1GPG) service  
     example [1247](#)  
 getpgrp (BPX4GPG) service  
     example [1340](#)  
 getpid (BPX1GPI, BPX4GPI) service [298](#)  
 getpid (BPX1GPI) service  
     example [1248](#)  
 getpid (BPX4GPI) service  
     example [1341](#)  
 getppid (BPX1GPP, BPX4GPP) service [299](#)  
 getppid (BPX1GPP) service  
     example [1248](#)  
 getppid (BPX4GPP) service  
     example [1342](#)  
 getpriority (BPX1GPY, BPX4GPY) service [300](#)  
 getpriority (BPX1GPY) service  
     example [1249](#)  
 getpriority (BPX4GPY) service  
     example [1342](#)  
 getpwent (BPX1GPE, BPX4GPE) service [302](#)  
 getpwent (BPX1GPE) service  
     example [1247](#)  
 getpwent (BPX4GPE) service  
     example [1341](#)  
 getpwnam (BPX1GPN, BPX4GPN) service [305](#)  
 getpwnam (BPX1GPN) service  
     example [1248](#)  
 getpwnam (BPX4GPN) service  
     example [1341](#)  
 getpwuid (BPX1GPU, BPX4GPU) service [308](#)  
 getpwuid (BPX1GPU) service  
     example [1249](#)  
 getpwuid (BPX4GPU) service  
     example [1342](#)  
 getrandom  
     obtain series of random bytes [310](#)  
 getrandom (BPX1GRD, BPX4GRD) service [310](#)  
 getrandom (BPX1GRD)  
     example [1223](#), [1250](#)  
 getrandom (BPX4GRD)  
     example [1317](#), [1343](#)  
 getrlimit (BPX1GRL, BPX4GRL) service [313](#)  
 getrlimit (BPX1GRL) service  
     example [1250](#)  
 getrlimit (BPX4GRL) service  
     example [1343](#)  
 getrusage (BPX1GRU, BPX4GRU) service [315](#)  
 getrusage (BPX1GRU) service  
     example [1250](#)  
 getrusage (BPX4GRU) service



- getrusage (BPX4GRU) service (*continued*)
  - example [1343](#)
- getsid (BPX1GES, BPX4GES) service [317](#)
- getsid (BPX1GES) service
  - example [1242](#)
- getsid (BPX4GES) service
  - example [1336](#)
- getsockname or getpeername (BPX1GNM, BPX4GNM)
  - service [319](#)
- getsockopt or setsockopt (BPX1OPT, BPX4OPT) service [322](#)
- getsockopt or setsockopt (BPX1OPT) service
  - example [1268](#)
- getsockopt or setsockopt (BPX4OPT) service
  - example [1360](#)
- getuid (BPX1GUI, BPX4GUI) service [329](#)
- getuid (BPX1GUI) service
  - example [1252](#)
- getuid (BPX4GUI) service
  - example [1345](#)
- getwd (BPX1GWD, BPX4GWD) service [330](#)
- getwd (BPX1GWD) service
  - example [1252](#)
- getwd (BPX4GWD) service
  - example [1345](#)
- givesocket (BPX1GIV, BPX4GIV) service [332](#)
- givesocket (BPX1GIV) service
  - example [1245](#)
- givesocket (BPX4GIV) service
  - example [1339](#)
- grant
  - access to subsidiary pseudoterminal device [336](#)
- grantpt (BPX1GPT, BPX4GPT) service [336](#)
- grantpt (BPX1GPT) service
  - example [1249](#)
- grantpt (BPX4GPT) service
  - example [1342](#)
- group
  - identify with process ID [777](#)
- group database
  - access
    - by group ID [268](#)
    - by group name [271](#)
    - sequentially [266](#), [765](#)
- group ID
  - effective
    - obtain [263](#)
    - set [758](#)
  - foreground process
    - obtain [931](#)
    - set [944](#)
  - process
    - obtain [297](#)
  - real
    - obtain [265](#)
  - set [762](#)
  - supplementary
    - obtain list and number [274](#), [276](#)
    - set list [766](#)
- group name
  - group database
    - access [271](#)
  - group members
    - information [271](#)

## H

- handling extended events in a debugged process [638](#)
- handling program check or abend in a debugged process [636](#)
- heavyweight thread (HWT)
  - terminating [1422](#)
- high-level language
  - ESPIE or ESTAE routine [1415](#)
  - signal interface [1413](#)
- host name
  - get
    - of an IP address [279](#)
    - get from a socket address [291](#)
    - get IP address and alias [282](#)
- HWT [1422](#)

## I

- I/O
  - control [1025](#)
  - flush buffer [923](#)
- ID
  - supplementary group
    - obtain list and number [276](#)
- identify
  - group with process ID [777](#)
- initial pthread-creating task (IPT) [1421](#), [1422](#)
- inotify\_add\_watch (BPX1IAW, BPX4IAW) ) service [338](#)
- inotify\_add\_watch service
  - example [1252](#)
- inotify\_add\_watch) service
  - example [1346](#)
- inotify\_init (BPX1INI, BPX4INI) ) service [341](#)
- inotify\_init(BPX1INI) service
  - example [1253](#), [1346](#)
- inotify\_rm\_watch (BPX1IRW, BPX4IRW)) service [344](#)
- inotify\_rm\_watch service
  - example [1254](#), [1346](#)
- installation exits [1435](#)
- interrupt
  - return last delivered [648](#)
- interrupt request block (IRB) [5](#)
- interrupt state
  - change and examine [610](#)
- interrupt type
  - change and examine [613](#)
- invoking a z/OS UNIX service [1](#)
- IP address
  - get
    - of a host name [282](#)
    - get for a service name or location [251](#)
    - get host name and alias [279](#)
- IPCSDumpAccess (BPXGMPTR, BPXGMPT4) service [346](#)
- IPCSDumpOpenClose service [352](#)
- IPT [1421](#)
- isatty (BPX1ITY) service
  - example [1254](#)
- isatty (BPX2ITY, BPX4ITY) service [359](#)
- isatty (BPX2ITY) service
  - example [1254](#)
- isatty (BPX4ITY) service
  - example [1347](#)

## K

- kernel
  - address space [1](#)
  - clean up resources [484](#)
- keyboard
  - navigation [1451](#)
  - PF keys [1451](#)
  - shortcut keys [1451](#)
- kill (BPX1KIL, BPX4KIL) service [360](#)
- kill (BPX1KIL) service
  - example [1254](#)
- kill (BPX4KIL) service
  - example [1347](#)

## L

- lchattr (BPX1LCR, BPX4LCR) service [371](#)
- lchattr (BPX1LCR) service
  - example [1255](#)
- lchattr (BPX4LCR) service
  - example [1348](#)
- lchown (BPX1LCO, BPX4LCO) service [380](#)
- lchown (BPX1LCO) service
  - example [1255](#)
- lchown (BPX4LCO) service
  - example [1348](#)
- link
  - create to a file [383](#)
  - external [183](#)
  - symbolic [183](#)
  - to callable services [1](#)
- link (BPX1LNK, BPX4LNK) service [383](#)
- link (BPX1LNK) service
  - example [1258](#)
- link (BPX4LNK) service
  - example [1351](#)
- linkage conventions
  - for callable services [3](#)
- linkage stub
  - linking to [2](#)
- linkat (BPX1LNA, BPX4LNA) service [387](#)
- linkat (BPX1LNA) service
  - example [1257](#)
- linkat (BPX4LNA) service
  - example [1350](#)
- listen (BPX1LSN, BPX4LSN) service [391](#)
- listen (BPX1LSN) service
  - example [1258](#)
- listen (BPX4LSN) service
  - example [1351](#)
- loadhfs (BPX1LOD, BPX4LOD) service [393](#)
- loadhfs (BPX1LOD) service
  - example [1257](#)
- loadhfs (BPX4LOD) service
  - example [1350](#)
- loadhfs extended (BPX1LDX, BPX4LDX) service [399](#)
- loadhfs extended (BPX1LDX) service
  - example [1255](#)
- loadhfs extended (BPX4LDX) service
  - example [1348](#)
- locking information [216](#)
- login
  - perform security-related services [364](#)

- lseek (BPX1LSK, BPX4LSK) service [406](#)
- lseek (BPX1LSK) service
  - example [1258](#)
- lseek (BPX4LSK) service
  - example [1351](#)
- lstat (BPX1LST, BPX4LST) service [409](#)
- lstat (BPX1LST) service
  - example [1258](#)
- lstat (BPX4LST) service
  - example [1351](#)

## M

- macro
  - mapping (31-bit) [1055](#)
  - mapping (64-bit) [1187](#)
- manipulating data in a debugged process [639](#)
- mapped megabyte area
  - create [412](#)
  - services
    - request [416](#)
- mapping
  - macro (31-bit) [1055](#)
  - macro (64-bit) [1187](#)
- mapping macro
  - BPXYACC [1055](#)
  - BPXYAIO [1055](#), [1187](#)
  - BPXYATT [1057](#)
  - BPXYAUDT [1060](#)
  - BPXYBRLK [1060](#)
  - BPXYCCA [1060](#), [1189](#)
  - BPXYCID [1061](#)
  - BPXYCLNP [1062](#)
  - BPXYCONS [1062](#)
  - BPXYCW [1069](#)
  - BPXYDCOR [1069](#), [1190](#)
  - BPXYDIRE [1074](#)
  - BPXYEFD [1075](#)
  - BPXYENFO [1075](#)
  - BPXYEPOL [1075](#)
  - BPXYERNO [1076](#)
  - BPXYFCTL [1076](#)
  - BPXYFDUM [1077](#)
  - BPXYFLK [1078](#)
  - BPXYFTYP [1078](#)
  - BPXYFUIO [1078](#)
  - BPXYGIDN [1079](#)
  - BPXYGIDS [1080](#)
  - BPXYGRD [1080](#)
  - BPXYINFY [1080](#)
  - BPXYINHE [1084](#), [1195](#)
  - BPXYIOC6 [1094](#), [1196](#)
  - BPXYIOCC [1085](#)
  - BPXYIOV [1098](#), [1199](#)
  - BPXYIPCP [1098](#)
  - BPXYIPCQ [1098](#), [1199](#)
  - BPXYITIM [1101](#), [1202](#)
  - BPXYMMG [1101](#), [1202](#)
  - BPXYMNTE [1104](#)
  - BPXYMODE [1106](#)
  - BPXYMSG [1107](#), [1204](#)
  - BPXYMSGF [1107](#)
  - BPXYMSGH [1109](#), [1205](#)
  - BPXYMSGX [1109](#)

mapping macro (*continued*)

- [BPXYMTM 1110](#)
- [BPXYOCRT 1111, 1206](#)
- [BPXYOEXT 1112](#)
- [BPXYOPNF 1113](#)
- [BPXYPCF 1114](#)
- [BPXYPEDB 1114](#)
- [BPXYPGPS 1116](#)
- [BPXYPGTH 1119](#)
- [BPXYPOE 1123](#)
- [BPXYPOLL 1124](#)
- [BPXYPPSD 1124, 1206](#)
- [BPXYPRLI 1126](#)
- [BPXYPTAT 1127](#)
- [BPXYPTRC 1127](#)
- [BPXYPTXL 1141, 1208](#)
- [BPXYRFIS 1141](#)
- [BPXYRLIM 1142, 1208](#)
- [BPXYRMON 1142](#)
- [BPXYSECI 1144](#)
- [BPXYSECO 1144](#)
- [BPXYSECT 1144](#)
- [BPXYSEEK 1144](#)
- [BPXYSEL 1145](#)
- [BPXYSELT 1145, 1209](#)
- [BPXYSEM 1146, 1209](#)
- [BPXYSFDL 1146](#)
- [BPXYSFPL 1147, 1210](#)
- [BPXYSHM 1147, 1210](#)
- [BPXYSIGH 1148](#)
- [BPXYSINF 1150, 1211](#)
- [BPXYSMC 1150](#)
- [BPXYSOCK 1151](#)
- [BPXYSSET 1160, 1211](#)
- [BPXYSSTF 1161](#)
- [BPXYSTAT 1162](#)
- [BPXYTCCP 1163](#)
- [BPXYTHDQ 1164](#)
- [BPXYTHLI 1165](#)
- [BPXYTIMS 1168](#)
- [BPXYTIOS 1169](#)
- [BPXYUTSN 1172](#)
- [BPXYWAST 1172](#)
- [BPXYWLM 1173, 1212](#)
- [BPXYWNSZ 1179](#)
- [BPXZOAPB 1179](#)
- [BPXZOCVT 1180](#)
- [BPXZOTCB 1181](#)

measure

- [resources 699](#)

mediumweight thread (MWT)

- [terminating 1422](#)

memory

- [map 435](#)
- [synchronization 469](#)
- [unmap pages 473](#)

memory map [435, 469](#)

memory mapping

- [protection of 452](#)

messages

- [send to the console 142](#)

mkdir (BPX1MKD, BPX4MKD) service [420](#)

mkdir (BPX1MKD) service

- [example 1260](#)

mkdir (BPX4MKD) service

- [example 1353](#)

mkdirat (BPX1MDA, BPX4MKD) service [423](#)

mkdirat (BPX1MDA) service

- [example 1259](#)

mkdirat (BPX4MDA) service

- [example 1352](#)

mknod (BPX1MKN, BPX4MKN) service [427](#)

mknod (BPX1MKN) service

- [example 1261](#)

mknod (BPX4MKN) service

- [example 1354](#)

mknodat (BPX1MNA, BPX4MNA) service [431](#)

mknodat (BPX1MNA) service

- [example 1262](#)

mknodat (BPX4MNA) service

- [example 1355](#)

mmap (BPX1MMP, BPX4MMP) service [435](#)

mmap (BPX1MMP) service

- [example 1262](#)

mmap (BPX4MMP) service

- [example 1355](#)

mode

- [change directory 101, 206](#)

- [change file 101, 206](#)

modification

- [set times for file 981](#)

module

- [invoking 2](#)

mount (BPX1MNT) service

- [example 1263](#)

mounted file system

- [information 995](#)

mprotect (BPX1MPR, BPX4MPR) service [452](#)

mprotect (BPX1MPR) service

- [example 1265](#)

mprotect (BPX4MPR) service

- [example 1357](#)

msgctl

- [message control operations 455](#)

msgctl (BPX1QCT, BPX4QCT) service [455](#)

msgctl (BPX1QCT) service

- [example 1276](#)

msgctl (BPX4QCT) service

- [example 1368](#)

msgget

- [get a message queue 458](#)

msgget (BPX1QGT, BPX4QGT) service [458](#)

msgget (BPX1QGT) service

- [example 1276](#)

msgget (BPX4QGT) service

- [example 1369](#)

msgrcv

- [message queue receive 462](#)

msgrcv (BPX1QRC, BPX4QRC) service [462](#)

msgrcv (BPX1QRC) service

- [example 1277](#)

msgrcv (BPX4QRC) service

- [example 1369](#)

msgsnd

- [message queue send. 465](#)

msgsnd (BPX1QSN, BPX4QSN) service [465](#)

msgsnd (BPX1QSN) service

- [example 1277](#)



- msgsnd (BPX4QSN) service
  - example [1370](#)
- msync (BPX1MSY, BPX4MSY) service [469](#)
- msync (BPX1MSY) service
  - example [1266](#)
- msync (BPX4MSY) service
  - example [1358](#)
- multiple address space
  - restrictions [6](#)
- multiple pthreads
  - terminating [1422](#)
- multiple task
  - signals created by ATTACH [1417](#)
- multiprocess debugging mode [641](#)
- multiprocess environment restrictions [6](#)
- munmap (BPX1MUN, BPX4MUN) service [473](#)
- munmap (BPX1MUN) service
  - example [1266](#)
- munmap (BPX4MUN) service
  - example [1358](#)
- MVS program
  - execution [58](#), [175](#)
- MVS signals
  - set up [487](#)
- mvsiptaffinity (BPX1IPT, BPX4IPT) service [476](#)
- mvspause (BPX1MP, BPX4MP) service [479](#)
- mvspause (BPX1MP) service
  - example [1263](#)
- mvspause (BPX4MP) service
  - example [1356](#)
- mvspauseinit (BPX1MPI, BPX4MPI) service [481](#)
- mvspauseinit (BPX1MPI) service
  - example [1265](#)
- mvspauseinit (BPX4MPI) service
  - example [1357](#)
- mvspocclp (BPX1MPC, BPX4MPC) service [484](#)
- mvspocclp (BPX1MPC) service
  - example [1264](#)
- mvspocclp (BPX4MPC) service
  - example [1356](#)
- mvssigsetup (BPX1MSS, BPX4MSS) service [487](#)
- mvssigsetup (BPX1MSS) service
  - example [1266](#)
- mvssigsetup (BPX4MSS) service
  - example [1358](#)
- MVSThreadAffinity (BPX1TAF, BPX4TAF) service [492](#)
- mvunsigsetup (BPX1MSD, BPX4MSD) service [495](#)
- mvunsigsetup (BPX1MSD) service
  - example [1265](#)
- mvunsigsetup (BPX4MSD) service
  - example [1357](#)
- MWT [1422](#)

## N

- navigation
  - keyboard [1451](#)
- nested callable services [5](#)
- nice (BPX1NIC, BPX4NIC) service [497](#)
- nice (BPX1NIC) service
  - example [1266](#)
- nice (BPX4NIC) service
  - example [1358](#)
- nonreentrant code [5](#)

## O

- obtain
  - effective group ID [263](#)
  - effective user ID [264](#)
  - file status [409](#), [893](#)
  - file system status [244](#), [896](#), [1029](#)
  - foreground process group ID [931](#)
  - group ID
    - process [297](#)
  - mounted file system
    - information [995](#)
  - path name
    - working directory [260](#)
  - pathname
    - working directory [330](#)
  - process data [999](#)
  - process ID
    - parent process [299](#)
  - real group ID [265](#)
  - real user ID [329](#)
  - series of random bytes [310](#)
  - supplementary group ID [276](#)
  - symbolic logic status information [409](#)
  - terminal attributes [926](#)
  - terminal name [956](#), [958](#)
  - user information by user ID [308](#)
  - user login name [289](#)
  - working directory
    - path name [260](#)
    - pathname [330](#)
- oe\_env\_np (BPX1ENV, BPX4ENV) service [500](#)
- oe\_env\_np (BPX1ENV) service
  - example [1233](#)
- oe\_env\_np (BPX4ENV) service
  - example [1326](#)
- offset
  - change file [406](#)
  - system control
    - callable services [1045](#)
- open
  - directory [523](#)
  - dump [352](#)
  - file and create descriptor [511](#)
  - open (BPX1OPN, BPX4OPN) service [511](#)
  - open (BPX1OPN) service
    - example [1267](#)
  - open (BPX2OPT) service
    - example [1267](#)
  - open (BPX4OPN) service
    - example [1359](#)
  - open (BPX4OPS) service
    - example [1359](#)
  - openat/openat2 (BPX1OAT, BPX4OAT) [518](#)
  - opendir (BPX1OPD, BPX4OPD) service [523](#)
  - opendir (BPX1OPD) service
    - example [1267](#)
  - opendir (BPX4OPD) service
    - example [1359](#)
  - openstat (BPX2OPN, BPX4OPS) service [526](#)
  - operating system
    - display name of current [964](#)
  - output
    - hold processing for transmission [918](#)

## P

- parameter
    - description [2](#), [4](#)
    - lists [4](#)
  - parent process
    - process ID
      - obtain [299](#)
  - path name
    - create symbolic link to [900](#)
    - working directory
      - obtain [260](#)
  - pathconf (BPX1PCF, BPX4PCF) service [535](#)
  - pathconf (BPX1PCF) service
    - example [1269](#)
  - pathconf (BPX4PCF) service
    - example [1361](#)
  - pathname
    - obtain terminal [956](#), [958](#)
    - resolve [677](#)
    - working directory
      - obtain [330](#)
  - pause (BPX1PAS, BPX4PAS) service [539](#)
  - pause (BPX1PAS) service
    - example [1269](#)
  - pause (BPX4PAS) service
    - example [1361](#)
  - PEDB (Process Exit Data Block) [1435](#)
  - permanent
    - write to [247](#)
  - PFS control [541](#)
  - pfscctl (BPX1PCT, BPX4PCT) service [541](#)
  - pfscctl (BPX1PCT) service
    - example [1269](#)
  - pfscctl (BPX4PCT) service
    - example [1361](#)
  - PID
    - affinity
      - process termination [548](#)
  - pipe
    - create [551](#)
    - create new instance [16](#)
  - pipe (BPX1PIP, BPX4PIP) service [551](#)
  - pipe (BPX1PIP) service
    - example [1270](#)
  - pipe (BPX4PIP) service
    - example [1362](#)
  - pipe2
    - create new instance [553](#)
  - pipe2 (BPX1PI2, BPX4PI2) service [553](#)
  - pipe2 service
    - example [1270](#), [1362](#)
  - pivot\_root (BPX1PVR) service
    - example [1275](#)
  - pivot\_root (BPX4PVR) service
    - example [1368](#)
  - poll (BPX1POL, BPX4POL) service [564](#)
  - poll (BPX1POL) service
    - example [1270](#)
  - poll (BPX4POL) service
    - example [1362](#)
  - port of entry information
    - specify [559](#)
  - postprocess initiation exit - (BPX\_POSPROC\_INIT) [1435](#)
  - prctl (BPX1PRC, BPX4PRC) service [568](#)
  - prctl (BPX1PRCL)
    - example [1271](#)
  - prctl (BPX4PRCL)
    - example [1363](#)
  - pread() and pwrite() (BPX1RW, BPX4RW) service [572](#)
  - preprocess initiation exit - (BPX\_PREPROC\_INIT) [1435](#)
  - preprocess termination exit - (BPX\_PREPROC\_TERM) [1435](#)
  - prlimit (BPX1PRL, BPX4PRL) service [574](#)
  - prlimit (BPX1PRL) service
    - example [1271](#), [1363](#), [1366](#)
  - process
    - cloning child [114](#)
    - control for debugging [620](#)
    - creating new [227](#)
    - end
      - bypass cleanup [180](#)
    - obtain data [325](#), [999](#)
    - obtain ID [298](#)
    - obtain time used [950](#)
    - parent
      - obtain process ID [299](#)
    - queue a signal to [848](#)
    - signal a [360](#)
    - signal mask
      - examine or change [845](#)
    - status of debugging [984](#)
    - suspend
      - pending a signal [539](#)
    - suspend execution [859](#)
  - Process Exit Data Block (PEDB) [1435](#)
  - process group
    - queue a signal to [848](#)
  - process group ID
    - for controlling terminal [933](#)
  - foreground
    - obtain [931](#)
    - set [944](#)
  - get for session leader [933](#)
  - obtain [297](#)
- process ID
  - identify group with [777](#)
  - obtain [298](#)
- parent process
  - obtain [299](#)
- process image initiation exit - (BPX\_IMAGE\_INIT) [1435](#)
- process start and end exits
  - postprocess initiation exit - (BPX\_POSPROC\_INIT) [1435](#)
  - preprocess initiation exit - (BPX\_PREPROC\_INIT) [1435](#)
  - preprocess termination exit - (BPX\_PREPROC\_TERM) [1435](#)
  - process image initiation exit - (BPX\_IMAGE\_INIT) [1435](#)
- process time
  - obtain [950](#)
- program
  - execution
    - MVS [58](#), [175](#)
  - execution on IPT [476](#)
- protection
  - of memory mapping [452](#)
- pseudoterminal
  - BPX1TSC, BPX4TSC (tcsetcp) [940](#)
  - flush I/O buffer [923](#)
  - get terminal code page [928](#)

- pseudoterminal (*continued*)
  - set attributes [937](#)
  - set terminal code page [940](#)
  - tcsetcp (BPX1TSC, BPX4TSC) [940](#)
- pthread
  - security
    - security environment [601](#)
  - tag
    - tag thread [615](#)
  - pthread\_cancel (BPX1PTB, BPX4PTB) service [579](#)
  - pthread\_cancel (BPX1PTB) service
    - example [1272](#)
  - pthread\_cancel (BPX4PTB) service
    - example [1364](#)
  - pthread\_create (BPX1PTC, BPX4PTC) service [581](#)
  - pthread\_create (BPX1PTC) service
    - example [1272](#)
  - pthread\_create (BPX4PTC) service
    - example [1364](#)
  - pthread\_create task initialization routine [1421](#)
  - pthread\_detach (BPX1PTD, BPX4PTD) service [586](#)
  - pthread\_detach (BPX1PTD) service
    - example [1273](#)
  - pthread\_detach (BPX4PTD) service
    - example [1365](#)
  - pthread\_exit\_and\_get (BPX1PTX, BPX4PTX) service [588](#)
  - pthread\_exit\_and\_get (BPX1PTX) service
    - example [1275](#)
  - pthread\_exit\_and\_get (BPX4PTX) service
    - example [1367](#)
  - pthread\_join (BPX1PTJ, BPX4PTJ) service [592](#)
  - pthread\_join (BPX1PTJ) service
    - example [1273](#)
  - pthread\_join (BPX4PTJ) service
    - example [1365](#)
  - pthread\_kill (BPX1PTK, BPX4PTK) service [595](#)
  - pthread\_kill (BPX1PTK) service
    - example [1274](#)
  - pthread\_kill (BPX4PTK) service
    - example [1366](#)
  - pthread\_quiesce (BPX1PTQ, BPX4PTQ) service [598](#)
  - pthread\_quiesce (BPX1PTQ) service
    - example [1274](#)
  - pthread\_quiesce (BPX4PTQ) service
    - example [1366](#)
  - pthread\_security\_np (BPX1TLS, BPX4TLS) service [601](#)
  - pthread\_security\_np (BPX1TLS) service
    - example [1304](#)
  - pthread\_security\_np (BPX4TLS) service
    - example [1396](#)
  - pthread\_self (BPX1PTS, BPX4PTS) service [609](#)
  - pthread\_self (BPX1PTS) service
    - example [1274](#)
  - pthread\_self (BPX4PTS) service
    - example [1367](#)
  - pthread\_setintr (BPX1PSI, BPX4PSI) service [610](#)
  - pthread\_setintr (BPX1PSI) service
    - example [1272](#)
  - pthread\_setintr (BPX4PSI) service
    - example [1364](#)
  - pthread\_setintrtype (BPX1PST, BPX4PST) service [613](#)
  - pthread\_setintrtype (BPX1PST) service
    - example [1272](#)
  - pthread\_setintrtype (BPX4PST) service

- pthread\_setintrtype (BPX4PST) service (*continued*)
  - example [1364](#)
- pthread\_tag\_np (BPX1PTT, BPX4PTT) service [615](#)
- pthread\_tag\_np (BPX1PTT) service
  - example [1275](#)
- pthread\_tag\_np (BPX4PTT) service
  - example [1367](#)
- pthread\_testintr (BPX1PTI, BPX4PTI) service [618](#)
- pthread\_testintr (BPX1PTI) service
  - example [1273](#)
- pthread\_testintr (BPX4PTI) service
  - example [1365](#)
- pthreads
  - callable services [1421](#)
  - create [1421](#)
  - terminating [1422](#)
- ptrace
  - status of process [984](#)
- ptrace (BPX1PTR, BPX4PTR) service [620](#)
- ptrace (BPX1PTR) service
  - example [1274](#)
- ptrace (BPX4PTR) service
  - example [1366](#)
- Pwrite (BPX1RW) service
  - example [1283](#)
- Pwrite (BPX4RW) service
  - example [1375](#)

## Q

- query
  - dub status [646](#)
- querydub (BPX1QDB, BPX4QDB) service [646](#)
- querydub (BPX1QDB) service
  - example [1276](#)
- querydub (BPX4QDB) service
  - example [1369](#)
- queue\_interrupt (BPX1SPB, BPX4SPB) service [648](#)
- queue\_interrupt (BPX1SPB) service
  - example [1292](#)
- queue\_interrupt (BPX4SPB) service
  - example [1385](#)
- quiesce
  - file system [650](#)
  - process [598](#)
- quiesce (BPX1QSE, BPX4QSE) service [650](#)
- quiesce (BPX1QSE) service
  - example [1277](#)
- quiesce (BPX4QSE) service
  - example [1370](#)
- quiesce process
  - threads [598](#)

## R

- read
  - directory entry [656](#), [659](#)
  - dump information [346](#)
  - external link value [663](#)
  - file [572](#), [652](#)
  - symbolic link value [666](#), [669](#)
- read (BPX1RED, BPX4RED) service [652](#)
- read (BPX1RED) service

read (BPX1RED) service (*continued*)  
     example [1280](#)  
 read (BPX4RED) service  
     example [1372](#)  
 read extlink (BPX1RDX) service  
     example [1279](#)  
 read extlink (BPX4RDX) service  
     example [1372](#)  
 read\_extlink (BPX1RDX, BPX4RDX) service [663](#)  
 readdir (BPX1RDD, BPX4RDD) service [656](#)  
 readdir (BPX1RDD) service  
     example [1278](#)  
 readdir (BPX4RDD) service  
     example [1371](#)  
 readdir2 (BPX1RD2, BPX4RD2) service [659](#)  
 readdir2 (BPX1RD2) service  
     example [1279](#)  
 readdir2 (BPX4RD2) service  
     example [1372](#)  
 readlink (BPX1RDL, BPX4RDL) service [666](#)  
 readlink (BPX1RDL) service  
     example [1278](#)  
 readlink (BPX1RLA, BPX4RDL) service [669](#)  
 readlink (BPX4RDL) service  
     example [1371](#)  
 readlinkat (BPX1RLA) service  
     example [1281](#)  
 readlinkat (BPX4RLA) service  
     example [1373](#)  
 readv (BPX1RDV, BPX4RDV) service [673](#)  
 readv (BPX1RDV) service  
     example [1279](#)  
 readv (BPX4RDV) service  
     example [1371](#)  
 real user ID  
     obtain [329](#)  
 realpath (BPX1RPH, BPX4RPH) service [677](#)  
 realpath (BPX1RPH) service  
     example [1283](#)  
 realpath (BPX4RPH) service  
     example [1375](#)  
 reason code  
     description [3](#)  
 receiving notification of events in a debugged process [634](#)  
 recv (BPX1RCV, BPX4RCV) service [680](#)  
 recv (BPX1RCV) service  
     example [1278](#)  
 recv (BPX4RCV) service  
     example [1370](#)  
 recvfrom (BPX1RFM, BPX4RFM) service [683](#)  
 recvfrom (BPX1RFM) service  
     example [1281](#)  
 recvfrom (BPX4RFM) service  
     example [1373](#)  
 recvmmsg (BPX2RMS, BPX4RMS) service [687](#)  
 recvmmsg (BPX2RMS) service  
     example [1282](#)  
 recvmmsg (BPX4RMS) service  
     example [1374](#)  
 reenterable code [1219](#), [1313](#)  
 reentrant code [5](#)  
 register interest in  
     file  
         by descriptor [1003](#)

register interest in (*continued*)  
     file (*continued*)  
         by path name [1025](#)  
 register usage  
     for callable services [3](#)  
 release level  
     determining [3](#)  
 remote-terminal  
     get terminal code page [928](#)  
     set terminal code page names and conversion tables  
         [946](#)  
 remote-TTY  
     get terminal code page [928](#)  
     set terminal code page names and conversion tables  
         [946](#)  
 remove  
     directory [703](#)  
     directory entry [966](#)  
     directory or directory entry [969](#)  
     virtual file system [961](#)  
 rename  
     directory [690](#)  
     file [690](#)  
 rename (BPX1REN, BPX4REN) service [690](#)  
 rename (BPX1REN) service  
     example [1280](#)  
 rename (BPX4REN) service  
     example [1373](#)  
 renameat (BPX1RNA, BPX4RNA) service [694](#)  
 renameat (BPX1RNA) service  
     example [1282](#)  
 renameat (BPX4RNA) service  
     example [1375](#)  
 reset directory to the beginning [701](#)  
 resolve  
     pathname [677](#)  
 resource (BPX1RMG, BPX4RMG) service [699](#)  
 resource (BPX1RMG) service  
     example [1281](#)  
 resource (BPX4RMG) service  
     example [1374](#)  
 Resource Access Control Facility (RACF) [8](#)  
 resources  
     clean up kernel [484](#)  
     measure [699](#)  
 restrictions, environmental [5](#)  
 restrictions, multiprocess [6](#)  
 resume  
     terminal data flow [921](#)  
 resuming or detaching from a debugged process [640](#)  
 return  
     file mode creation mask [960](#)  
     last interrupt delivered [648](#)  
 return code  
     description [2](#)  
 return value  
     description [2](#)  
 rewind directory to the beginning [701](#)  
 rewinddir (BPX1RWD, BPX4RWD) service [701](#)  
 rewinddir (BPX1RWD) service  
     example [1283](#)  
 rewinddir (BPX4RWD) service  
     example [1376](#)  
 rmdir (BPX1RMD, BPX4RMD) service [703](#)

- rmdir (BPX1RMD) service
  - example [1281](#)
- rmdir (BPX4RMD) service
  - example [1374](#)
- root directory
  - change [111](#)
- RTL (runtime library)
  - signals [1413](#)
- runtime library (RTL)
  - signals [1413](#)

## S

- security product
  - interface to [364](#)
- select (BPX1SEL, BPX4SEL) service [706](#)
- select (BPX1SEL) service
  - example [1285](#)
- select (BPX4SEL) service
  - example [1378](#)
- semctl
  - semaphore control operations [713](#)
- semctl (BPX1SCT, BPX4SCT) service [713](#)
- semctl (BPX1SCT) service
  - example [1284](#)
- semctl (BPX4SCT) service
  - example [1376](#)
- semget
  - get set of semaphores [718](#)
- semget (BPX1SGT, BPX4SGT) service [718](#)
- semget (BPX1SGT) service
  - example [1287](#)
- semget (BPX4SGT) service
  - example [1380](#)
- semop
  - semaphore operations [723](#)
- semop (BPX1SOP, BPX4SOP) service [723](#)
- semop (BPX1SOP) service
  - example [1292](#)
- semop (BPX4SOP) service
  - example [1384](#)
- send
  - messages to the console [142](#)
- send (BPX1SND, BPX4SND) service [726](#)
- send (BPX1SND) service
  - example [1291](#)
- send (BPX4SND) service
  - example [1383](#)
- send a signal [360](#)
- send\_file (BPX1SF, BPX4SF) service [730](#)
- send\_file (BPX1SF) service
  - example [1286](#)
- send\_file (BPX4SF) service
  - example [1379](#)
- sendmsg (BPX2SMS, BPX4SMS) service [734](#)
- sendmsg (BPX2SMS) service
  - example [1290](#)
- sendmsg (BPX4SMS) service
  - example [1383](#)
- sendto (BPX1STO, BPX4STO) service [738](#)
- sendto (BPX1STO) service
  - example [1298](#)
- sendto (BPX4STO) service
  - example [1391](#)

- serial data
  - break transmission of asynchronous [934](#)
- server
  - process work [746](#)
- server\_init (BPX1SIN, BPX4SIN) service [742](#)
- server\_init (BPX1SIN) service
  - example [1288](#)
- server\_init (BPX4SIN) service
  - example [1381](#)
- server\_pwu (BPX1SPW, BPX4SPW) service [746](#)
- server\_pwu (BPX1SPW) service
  - example [1294](#)
- server\_pwu (BPX4SPW) service
  - example [1387](#)
- service location
  - get
    - IP address [251](#)
- service name
  - get
    - IP address [251](#)
    - get from a socket address [291](#)
- session
  - create
    - set process group ID [793](#)
- set
  - effective group ID [758](#)
  - effective user ID [760](#)
  - file access time [981](#)
  - file mode creation mask [960](#)
  - file modification time [981](#)
  - foreground process group ID [944](#)
  - group ID [762](#)
  - terminal attributes [937](#)
  - terminal code page [940](#)
  - terminal code page names and conversion tables [946](#)
  - thread limits [795](#)
  - timer event [798](#)
  - user ID [800](#)
- set an alarm [30](#)
- set up
  - MVS signals [487](#)
- set\_dub\_default (BPX1SDD, BPX4SDD) service [752](#)
- set\_thread\_limits (BPX1STL, BPX4STL) service [795](#)
- set\_thread\_limits (BPX1STL) service
  - example [1298](#)
- set\_thread\_limits (BPX4STL) service
  - example [1390](#)
- set\_timer\_event (BPX1STE, BPX4STE) service [798](#)
- set\_timer\_event (BPX1STE> service
  - example [1297](#)
- set\_timer\_event (BPX4STE> service
  - example [1390](#)
- setdubdefault (BPX1SDD) service
  - example [1284](#)
- setdubdefault (BPX4SDD) service
  - example [1377](#)
- setegid (BPX1SEG, BPX4SEG) service [758](#)
- setegid (BPX1SEG) service
  - example [1285](#)
- setegid (BPX4SEG) service
  - example [1378](#)
- seteuid (BPX1SEU, BPX4SEU) service [760](#)
- seteuid (BPX1SEU) service
  - example [1286](#)

- seteuid (BPX4SEU) service
  - example [1378](#)
- setgid (BPX1SGI, BPX4SGI) service [762](#)
- setgid (BPX1SGI) service
  - example [1287](#)
- setgid (BPX4SGI) service
  - example [1379](#)
- setgrent (BPX1SGE, BPX4SGE) service [765](#)
- setgrent (BPX1SGE) service
  - example [1286](#)
- setgrent (BPX4SGE) service
  - example [1379](#)
- setgroups (BPX1SGR, BPX4SGR) service [766](#)
- setgroups (BPX1SGR) service
  - example [1287](#)
- setgroups (BPX4SGR) service
  - example [1380](#)
- setitimer (BPX1STR, BPX4STR) service [768](#)
- setitimer (BPX1STR) service
  - example [1298](#)
- setitimer (BPX4STR) service
  - example [1391](#)
- setns (BPX1SNS, BPX4SNS) service [771](#)
- setns (BPX1SNS) service
  - example [1291](#)
- setns (BPX4SNS) service
  - example [1384](#)
- setpeer (BPX1SPR, BPX4SPR) service [775](#)
- setpeer (BPX1SPR) service
  - example [1294](#)
- setpeer (BPX4SPR) service
  - example [1387](#)
- setpgid (BPX1SPG, BPX4SPG) service [777](#)
- setpgid (BPX1SPG) service
  - example [1293](#)
- setpgid (BPX4SPG) service
  - example [1385](#)
- setpriority (BPX1SPY, BPX4SPY) service [779](#)
- setpriority (BPX1SPY) service
  - example [1295](#)
- setpriority (BPX4SPY) service
  - example [1387](#)
- setpwent (BPX1SPE, BPX4SPE) service [782](#)
- setpwent (BPX1SPE) service
  - example [1292](#)
- setpwent (BPX4SPE) service
  - example [1385](#)
- setregid (BPX1SRG, BPX4SRG) service [784](#)
- setregid (BPX1SRG) service
  - example [1295](#)
- setregid (BPX4SRG) service
  - example [1388](#)
- setreuid (BPX1SRU, BPX4SRU) service [786](#)
- setreuid (BPX1SRU) service
  - example [1296](#)
- setreuid (BPX4SRU) service
  - example [1388](#)
- setrlimit (BPX1SRL, BPX4SRL) service [789](#)
- setrlimit (BPX1SRL) service
  - example [1295](#)
- setrlimit (BPX4SRL) service
  - example [1388](#)
- setsid (BPX1SSI, BPX4SSI) service [793](#)
- setsid (BPX1SSI) service

- setsid (BPX1SSI) service (*continued*)
  - example [1296](#)
- setsid (BPX4SSI) service
  - example [1389](#)
- setting a breakpoint in a debugged process [639](#)
- setuid (BPX1SUI, BPX4SUI) service [800](#)
- setuid (BPX1SUI) service
  - example [1299](#)
- setuid (BPX4SUI) service
  - example [1392](#)
- setup
  - linking to callable services for signals [1414](#)
- shmat
  - shared memory attach operation [804](#)
- shmat (BPX1MAT, BPX4MAT) service [804](#)
- shmat (BPX1MAT) service
  - example [1259](#)
- shmat (BPX4MAT) service
  - example [1352](#)
- shmctl
  - shared memory control operations [809](#)
- shmctl (BPX1MCT, BPX4MCT) service [809](#)
- shmctl (BPX1MCT) service
  - example [1259](#)
- shmctl (BPX4MCT) service
  - example [1352](#)
- shmdt
  - detach shared memory segment [812](#)
- shmdt (BPX1MDT, BPX4MDT) service [812](#)
- shmdt (BPX1MDT) service
  - example [1260](#)
- shmdt (BPX4MDT) service
  - example [1353](#)
- shmем\_lock (BPX1SLK, BPX4SLK) service [814](#)
- shmем\_lock (BPX1SLK) service
  - example [1289](#)
- shmем\_lock (BPX4SLK) service
  - example [1382](#)
- shmем\_mutex\_condvar (BPX1SMC, BPX4SMC) service [819](#)
- shmget
  - get shared memory segment [827](#)
- shmget (BPX1MGT, BPX4MGT) service [827](#)
- shmget (BPX1MGT) service
  - example [1260](#)
- shmget (BPX4MGT) service
  - example [1353](#)
- shortcut keys [1451](#)
- shutdown (BPX1SHT, BPX4SHT) service [832](#)
- shutdown (BPX1SHT) service
  - example [1288](#)
- shutdown (BPX4SHT) service
  - example [1380](#)
- sigaction (BPX1SIA, BPX4SIA) service [834](#)
- sigaction (BPX1SIA) service
  - example [1288](#)
- sigaction (BPX4SIA) service
  - example [1381](#)
- signal
  - deferral [1416](#)
  - delayed delivery [1416](#)
  - delivery keys [1416](#)
  - ESPIE or ESTAE macro, with [1415](#)
  - examine pending [843](#)
  - queue to a process [848](#)



- signal (*continued*)
  - queue to a process group [848](#)
  - runtime library (RTL) [1413](#)
  - set up for MVS [487](#)
  - setup for linking to callable services [1414](#)
  - suspend a process
    - pending a signal [539](#)
  - wait for
    - with a specified timeout [854](#)
- signal action
  - change or examine [834](#)
- signal actions
  - change [839](#)
  - examine [839](#)
- signal delivery
  - disable [899](#)
  - enable [899](#)
- signal interface
  - high-level language [1413](#)
- signal interface routine (SIR) [1413](#)
- signal mask
  - change [851](#)
  - examine or change [845](#)
- signal setup
  - detach [495](#)
- signals
  - relationship to callable services [1413](#)
  - services supported with [1413](#)
- sigpending (BPX1SIP, BPX4SIP) service [843](#)
- sigpending (BPX1SIP) service
  - example [1289](#)
- sigpending (BPX4SIP) service
  - example [1381](#)
- sigprocmask (BPX1SPM, BPX4SPM) service [845](#)
- sigprocmask (BPX1SPM) service
  - example [1293](#)
- sigprocmask (BPX4SPM) service
  - example [1385](#)
- sigqueue (BPX1SGQ, BPX4SGQ) service [848](#)
- sigqueue (BPX1SGQ) service
  - example [1287](#)
- sigqueue (BPX4SGQ) service
  - example [1379](#)
- sigsuspend (BPX1SSU, BPX4SSU) service [851](#)
- sigsuspend (BPX1SSU) service
  - example [1297](#)
- sigsuspend (BPX4SSU) service
  - example [1389](#)
- sigtimedwait (BPX1STW, BPX4STW) service [854](#)
- sigtimedwait (BPX1STW) service
  - example [1299](#)
- sigtimedwait (BPX4STW) service
  - example [1392](#)
- sigwait (BPX1SWT, BPX4SWT) service [857](#)
- sigwait (BPX1SWT) service
  - example [1300](#)
- sigwait (BPX4SWT) service
  - example [1392](#)
- SIR (signal interface routine) [1413](#)
- sleep (BPX1SLP, BPX4SLP) service [859](#)
- sleep (BPX1SLP) service
  - example [1289](#)
- sleep (BPX4SLP) service
  - example [1382](#)
- smf\_record (BPX1SMF, BPX4SMF) service [861](#)
- smf\_record (BPX1SMF) service
  - example [1290](#)
- smf\_record (BPX4SMF) service
  - example [1382](#)
- socket
  - accept a connection [19](#)
  - send a file on [730](#)
- socket address
  - get host name from [291](#)
  - get service name from [291](#)
- socket connection
  - accept [19](#)
- socket or socketpair (BPX1SOC, BPX4SOC) service [865](#)
- socket or socketpair (BPX1SOC) service
  - example [1291](#)
- socket or socketpair (BPX4SOC) service
  - example [1384](#)
- spawn (BPX1SPN, BPX4SPN) service [869](#)
- spawn (BPX1SPN) service
  - example [1293](#)
- spawn (BPX4SPN) service
  - example [1386](#)
- SRB mode routines
  - callable services available to [1041](#)
  - supported in 31-bit mode [1042](#)
  - supported in 64-bit mode [1042](#)
- srx\_np (BPX1SRX, BPX4SRX) service [886](#)
- srx\_np (BPX1SRX) service
  - example [1296](#)
- srx\_np (BPX4SRX) service
  - example [1389](#)
- starting [633](#)
- starting in debugging mode [633](#)
- stat (BPX1STA, BPX4STA) service [893](#)
- stat (BPX1STA) service
  - example [1297](#)
- stat (BPX4STA) service
  - example [1389](#)
- status
  - obtain file
    - by descriptor [238](#)
  - obtain file system [244](#), [896](#), [1029](#)
  - query dub [646](#)
- status information
  - obtain file [409](#), [893](#)
  - obtain symbolic link [409](#)
- statvfs (BPX1STV, BPX4STV) service [896](#)
- statvfs (BPX1STV) service
  - example [1299](#)
- statvfs (BPX4STV) service
  - example [1391](#)
- storage
  - permanent
    - write to [247](#)
- summary of changes [xxxiii](#)
- superuser [8](#)
- SUPERUSER.SHMMCV.LIMIT [824](#)
- supplementary group ID
  - obtain list and number [276](#)
- suspend
  - process execution [859](#)
  - terminal data flow [921](#)
- suspend processing

- suspend processing (*continued*)
  - for output transmission [918](#)
- sw\_sigdlv (BPX1DSD, BPX4DSD) service [899](#)
- symbolic link
  - create relative to directory file descriptor [905](#)
  - create to external name [183](#)
  - create to path name [900](#)
  - obtain status information [409](#)
  - read value [666](#), [669](#)
  - remove from directory [966](#)
  - remove from directory or directory entry [969](#)
- symlink (BPX1SYM, BPX4SYM) service [900](#)
- symlink (BPX1SYM) service
  - example [1301](#)
- symlink (BPX4SYM) service
  - example [1393](#)
- symlinkat (BPX1SYA, BPX4SYA) service [905](#)
- symlinkat service
  - example [1300](#), [1392](#)
- sync (BPX1SYN, BPX4SYN) service [910](#)
- sync (BPX1SYN) service
  - example [1301](#)
- sync (BPX4SYN) service
  - example [1394](#)
- syncfs (BPX1SYF, BPX4SYF) service [912](#)
- syncfs (BPX1SYF) service
  - example [1300](#)
- syncfs (BPX4SYF) service
  - example [1393](#)
- syntax
  - for z/OS UNIX callable services [1](#)
- SYS1.CSSLIB [2](#)
- syscall exits
  - enabling [1439](#)
- syscalls.h C header [1445](#)
- sysconf (BPX1SYC, BPX4SYC) service [913](#)
- sysconf (BPX1SYC) service
  - example [1300](#)
- sysconf (BPX4SYC) service
  - example [1393](#)
- sysplex
  - BPX2MNT, BPX4MNT (`__mount`) service [447](#)
  - getmntent (BPX1GMN, BPX4GMN) service [995](#)
  - quiesce restrictions [650](#)
  - umount callable service [961](#)
- SYSSTATE macro [8](#)
- system configuration
  - options [913](#)
- system control
  - offsets to callable services [1045](#)

## T

- takesocket (BPX1TAK, BPX4TAK) service [916](#)
- takesocket (BPX1TAK) service
  - example [1301](#)
- takesocket (BPX4TAK) service
  - example [1394](#)
- tasks
  - terminating [1422](#)
- tcdrain (BPX1TDR, BPX4TDR) service [918](#)
- tcdrain (BPX1TDR) service
  - example [1302](#)

- tcdrain (BPX4TDR) service
  - example [1394](#)
- tcflow (BPX1TFW, BPX4TFW) service [921](#)
- tcflow (BPX1TFW) service
  - example [1302](#)
- tcflow (BPX4TFW) service
  - example [1395](#)
- tcflush (BPX1TFH, BPX4TFH) service [923](#)
- tcflush (BPX1TFH) service
  - example [1302](#)
- tcflush (BPX4TFH) service
  - example [1395](#)
- tcgetattr (BPX1TGA, BPX4TGA) service [926](#)
- tcgetattr (BPX1TGA) service
  - example [1302](#)
- tcgetattr (BPX4TGA) service
  - example [1395](#)
- tcgetcp (BPX1TGC, BPX4TGC) service [928](#)
- tcgetcp (BPX1TGC) service
  - example [1303](#)
- tcgetcp (BPX4TGC) service
  - example [1395](#)
- tcgetpgrp (BPX1TGP, BPX4TGP) service [931](#)
- tcgetpgrp (BPX1TGP) service
  - example [1303](#)
- tcgetpgrp (BPX4TGP) service
  - example [1396](#)
- tcgetsid (BPX1TGS, BPX4TGS) service [933](#)
- tcgetsid (BPX1TGS) service
  - example [1303](#)
- tcgetsid (BPX4TGS) service
  - example [1396](#)
- tcsendbreak (BPX1TSB, BPX4TSB) service [934](#)
- tcsendbreak (BPX1TSB) service
  - example [1305](#)
- tcsendbreak (BPX4TSB) service
  - example [1397](#)
- tcsetattr (BPX1TSA, BPX4TSA) service [937](#)
- tcsetattr (BPX1TSA) service
  - example [1304](#)
- tcsetattr (BPX4TSA) service
  - example [1397](#)
- tcsetcp (BPX1TSC, BPX4TSC) service [940](#)
- tcsetcp (BPX1TSC) service
  - example [1305](#)
- tcsetcp (BPX4TSC) service
  - example [1397](#)
- tcsetpgrp (BPX1TSP, BPX4TSP) service [944](#)
- tcsetpgrp (BPX1TSP) service
  - example [1305](#)
- tcsetpgrp (BPX4TSP) service
  - example [1398](#)
- tcsettables (BPX1TST, BPX4TST) service [946](#)
- tcsettables (BPX1TST) service
  - example [1305](#)
- tcsettables (BPX4TST) service
  - example [1398](#)
- terminal
  - break asynchronous serial data transmission [934](#)
  - flush I/O buffer [923](#)
  - get code page [928](#)
  - obtain attributes [926](#)
  - obtain name [956](#), [958](#)
  - set attributes [937](#)



- terminal (*continued*)
  - set code page [940](#)
  - set code page names and conversion tables [946](#)
- terminal data flow
  - suspend or resume [921](#)
- terminate
  - process
    - bypass cleanup [180](#)
- terminating
  - heavyweight thread (HWT) [1422](#)
  - mediumweight thread (MWT) [1422](#)
  - multiple pthreads [1422](#)
  - pthreads [1422](#)
  - tasks [1422](#)
- termios data area [926](#)
- thread
  - cancel [579](#)
  - create [581](#)
- threads
  - callable services [1421](#)
  - create [1421](#)
- timer event
  - set [798](#)
- times (BPX1TIM, BPX4TIM) service [950](#)
- times (BPX1TIM) service
  - example [1303](#)
- times (BPX4TIM) service
  - example [1396](#)
- transmission
  - break for asynchronous serial data [934](#)
- transmission output
  - hold processing for [918](#)
- truncate
  - file [249](#), [953](#)
- truncate (BPX1TRU, BPX4TRU) service [953](#)
- truncate (BPX1TRU) service
  - example [1304](#)
- truncate (BPX4TRU) service
  - example [1397](#)
- ttynname (BPX1TYN, BPX4TYN) service [956](#)
- ttynname (BPX1TYN) service
  - example [1306](#)
- ttynname (BPX2TYN, BPX4TYN) service [958](#)
- ttynname (BPX2TYN) service
  - example [1306](#)
- ttynname (BPX4TYN) service
  - example [1399](#)

## U

- umask (BPX1UMK, BPX4UMK) service [960](#)
- umask (BPX1UMK) service
  - example [1307](#)
- umask (BPX4UMK) service
  - example [1399](#)
- umount (BPX1UMT, BPX4UMT) service [961](#)
- umount (BPX1UMT) service
  - example [1307](#)
- umount (BPX4UMT) service
  - example [1399](#)
- uname (BPX1UNA, BPX4UNA) service [964](#)
- uname (BPX1UNA) service
  - example [1307](#)
- uname (BPX4UNA) service

- uname (BPX4UNA) service (*continued*)
  - example [1400](#)
- undub [1](#)
- unlink (BPX1ULA) service
  - example [1306](#)
- unlink (BPX1UNL, BPX4UNL) service [966](#)
- unlink (BPX1UNL) service
  - example [1308](#)
- unlink (BPX4UNL) service
  - example [1400](#)
- unlinkat (BPX1ULA, BPX4ULA) service [969](#)
- unlinkat (BPX1ULA) service
  - example [1399](#)
- unlock
  - pseudoterminal manager and subsidiary pair [973](#)
- unlockpt (BPX1UPT, BPX4UPT) service [973](#)
- unlockpt (BPX1UPT) service
  - example [1308](#)
- unlockpt (BPX4UPT) service
  - example [1400](#)
- unmap
  - previously mapped pages [473](#)
- unquiesce
  - file system [975](#)
- unquiesce (BPX1UQS, BPX4UQS) service [975](#)
- unquiesce (BPX1UQS) service
  - example [1308](#)
- unquiesce (BPX4UQS) service
  - example [1400](#)
- unshare (BPX1UNS, BPX4UNS) service [977](#)
- unshare service (BPX1UNS)
  - example [1308](#)
- unshare service (BPX4UNS)
  - example [1401](#)
- user area description [642](#)
- user database
  - access
    - sequentially [302](#), [782](#)
    - user name [305](#), [530](#)
- user ID
  - obtain effective [264](#)
  - set [800](#)
  - set effective [760](#)
- user interface
  - ISPF [1451](#)
  - TSO/E [1451](#)
- user login name
  - obtain [289](#)
- utime (BPX1UTI, BPX4UTI) service [981](#)
- utime (BPX1UTI) service
  - example [1309](#)
- utime (BPX4UTI) service
  - example [1401](#)

## V

- virtual file system
  - remove [961](#)

## W

- w\_getipc
  - interprocess communications [991](#)

- w\_getipc (BPX1GET, BPX4GET) service [991](#)
- w\_getipc (BPX1GET) service
  - example [1242](#)
- w\_getipc (BPX4GET) service
  - example [1336](#)
- w\_getmntent (BPX1GMN, BPX4GMN) service [995](#)
- w\_getmntent (BPX1GMN) service
  - example [1246](#)
- w\_getmntent (BPX4GMN) service
  - example [1340](#)
- w\_getpsent (BPX1GPS) service
  - example [1248](#)
- w\_ioctl (BPX1IOC, BPX4IOC) service [1003](#)
- w\_ioctl (BPX1IOC) service
  - example [1253](#)
- w\_ioctl (BPX4IOC) service
  - example [1347](#)
- w\_pioctl (BPX1PIO, BPX4PIO) service [1025](#)
- w\_statvfs (BPX1STF, BPX4STF) service [1029](#)
- w\_statvfs (BPX1STF) service
  - example [1297](#)
- w\_statvfs (BPX4STF) service
  - example [1390](#)
- wait
  - for a signal
    - with a specified timeout [854](#)
  - for asynchronous I/O request [27](#)
  - user events and signal event [481](#)
  - user events plus signals [479](#)
- wait (BPX1WAT, BPX4WAT) service [984](#)
- wait (BPX1WAT) service
  - example [1309](#)
- wait (BPX4WAT) service
  - example [1401](#)
- wait extension (BPX1WTE) service
  - example [1310](#)
- wait extension (BPX4WTE) service
  - example [1403](#)
- wait-extension (BPX1WTE, BPX4WTE) service [987](#)
- working directory
  - change [98](#), [202](#)
  - path name
    - obtain [260](#)
  - pathname
    - obtain [330](#)
- working with threads in a debugged process [635](#)
- write
  - from a buffer to a file [1032](#)
  - permanent storage [247](#)
  - to a file [572](#)
- write (BPX1WRT, BPX4WRT) service [1032](#)
- write (BPX1WRT) service
  - example [1310](#)
- write (BPX4WRT) service
  - example [1402](#)
- writew (BPX1WRV, BPX4WRV) service [1036](#)
- writew (BPX1WRV) service
  - example [1310](#)
- writew (BPX4WRV) service
  - example [1402](#)

## Z

z/OS UNIX

z/OS UNIX (*continued*)

- accessing [1](#)
- connecting to [1](#)
- disconnecting from [1](#)
- dynamic syscall exits [1439](#)
- enabling [1439](#)
- exit points for syscalls [1439](#)
- process start and end exits [1435](#)
- threads [1421](#)
- threads [1421](#)





Product Number: 5655-ZOS

SA23-2281-70

