z/OS

UNIX System Services Planning: APAR OW54653 and OW54824

z/OS

UNIX System Services Planning: APAR OW54653 and OW54824

Contents

About this document

This document supports APARs OW54824 and OW54653 for UNIX System Services (z/OS UNIX), which are available for z/OS Version 1 Release 2.This document is available only on the z/OS UNIX System Services Web site at:

`http://www.ibm.com/servers/eserver/zseries/zos/unix/ow54824.html`

Chapter 1. Shared HFS in a Sysplex

This chapter describes shared HFS capability available as of OS/390 UNIX V2R9 for those who participate in a multi-system sysplex. It assumes that you already have a sysplex up. It will define what shared HFS is, introduce you to HFS data sets that exist in a sysplex, and help you establish that environment. The topics in this chapter reflect the tasks you must do.

Task	Topic
Creating the Sysplex Root HFS Data Set	7
Creating the System-Specific HFS Data Sets	8
Mounting the Version HFS	8
Creating an OMVS Couple Data Set (CDS)	10
Updating COUPLExx to Define the OMVS CDS to XCF	13
Customizing BPXPRMxx for Shared HFS	13
Keeping Automount Policies Consistent on All Systems in the Sysplex	23
Tuning z/OS UNIX Performance in a Sysplex	30

Although IBM recommends that you exploit shared HFS support, you are not required to. If you choose not to, you will continue to share HFS data sets as you have before OS/390 UNIX V2R9. To see how your file system structure differs in OS/390 UNIX V2R9 from V2R8, see “Comparing File Systems in Single System Pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or Later Environments” on page 3.

z/OS Parallel Sysplex Test Report describes how IBM’s integration test team implemented shared HFS.

What Does Shared HFS Mean?

Sysplex users can access data throughout the file hierarchy.

The best way to describe the benefit of this function is by comparing what was the file system sharing capability prior to OS/390 UNIX V2R9 with the capability that exists now. Consider a sysplex that consists of two systems, SY1 and SY2:

- Users logged onto SY1 can write to the directories on SY1. For users on SY1 to make a change to file systems mounted on SY2’s **/u** directory, they would have to log onto SY2.
- The system programmer who makes configuration changes for the sysplex needs to change the entries in the **/etc** file systems for SY1 and SY2. To make the changes for both systems, the system programmer must log onto each system.

With shared HFS, all file systems that are mounted by a system participating in shared HFS are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a OS/390 UNIX V2R9 sysplex that consists of two systems, SY1 and SY2:

- A user logged onto any system can make changes to file systems mounted on **/u**, and those changes are visible to all systems.

- The system programmer who manages maintenance for the sysplex can change entries in both **/etc** file systems from either system.

In this chapter, the term *participating group* is used to identify those systems that belong to the same SYSBPX XCF sysplex group and have followed the required installation and migration activities to participate in shared HFS. To be in the participating group, the system level must be at OS/390 UNIX V2R9 or later. Systems earlier than OS/390 UNIX V2R9 can coexist in the sysplex with systems using shared HFS support, but the earlier systems will only be able to share file systems mounted on other systems in read-only mode, and not in read/write mode.

With shared HFS, there is greater availability of data in case of system outage. There is also greater flexibility for data placement and the ability for a single BPXPRMxx member to define all the file systems in the sysplex.

How the End User Views the HFS

This chapter describes the kinds of file systems and data sets that support the shared HFS capability in the sysplex. Figure 1 shows that, to the end users, the logical view of the HFS does not change for OS/390 UNIX V2R9. From their point of view, accessing files and directories in the system is just the same. That is true for all end users, whether they are in a sysplex or not.

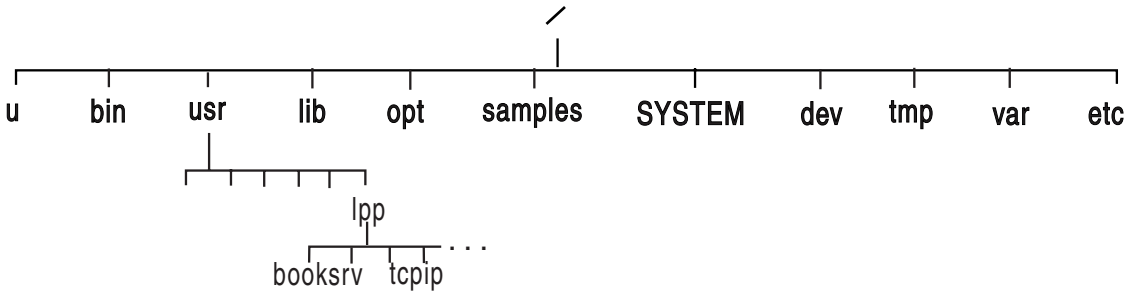

Figure 1. Logical View of Shared HFS for the End User

This logical view applies to the end user only. However, system programmers need to know that the illustration of directories found in Figure 1 does not reflect the physical view of file systems. Starting in OS/390 UNIX V2R9, some of the directories are actually symbolic links, as is described in the following information.

Summary of New HFS Data Sets

This chapter introduces HFS data sets and terms needed to use shared HFS. Table 1 on page 3 summarizes the HFS data sets that are needed in a sysplex that has shared HFS. As you study the illustrations of file system configurations in this chapter, you can refer back to this table.

Table 1. HFS Data Sets That Exist in a Sysplex

Name	Characteristics	Purpose	How Created
Sysplex root	It contains directories and symbolic links that allow redirection of directories. Only one sysplex root HFS is allowed for all systems participating in shared HFS.	The sysplex root is used by the system to redirect addressing to other directories. It is very small and is mounted read/write. See for a more complete description of the sysplex root HFS.	The user runs the BPXISYSR job.
System specific	It contains data specific to each system, including the **/dev**, **/tmp**, **/var**, and **/etc** directories for one system. There is one system-specific HFS data set for each system participating in the shared HFS capability.	The system-specific HFS data set is used by the system to mount system-specific data. It contains the necessary mount points for system-specific data and the symbolic links to access sysplex-wide data, and should be mounted read/write. See "Creating the System-Specific HFS Data Sets" on page 8 for a complete description of the system-specific HFS.	The user runs the BPXISYSS job on each participating system.
Version In a sysplex, *version HFS* is the new name for the root HFS.	It contains system code and binaries, including the **/bin**, **/usr**, **/lib**, **/opt**, and **/samples** directories. IBM delivers only one version root; you might define more as you add new system levels and new maintenance levels.	The version HFS has the same purpose as the root HFS in the non-sysplex world. It should be mounted read-only. See "Mounting the Version HFS" on page 8 for a complete description of the version HFS.	IBM supplies this HFS in the ServerPac. CBPDO users create the HFS by following steps defined in the Program Directory.

Comparing File Systems in Single System Pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or Later Environments

The illustrations in this section show you how the file system structures that existed before OS/390 UNIX V2R9 compare with the structures in OS/390 UNIX V2R9 and later. IBM's recommendations for several releases prior to OS/390 UNIX V2R9 has been that you separate the system setup parameters from the file system parameters so that each system in the sysplex has two BPXPRMxx members: a system limits member and a file system member. In the shared HFS environment, that separation of system limit parameters from file system parameters is even more important. In the shared HFS environment, each system will continue to have a system limits BPXPMRxx member. As you will see in sections that follow, with shared HFS, you might have a file system BPXPRMxx member for each participating system or you might replace those individual file system BPXPRMxx members with a single file system BPXPRMxx member for all participating systems.

File Systems in Single System Pre-OS/390 UNIX V2R9 Environments

The following example shows what BPXPRMxx file system parameters would look like in a single system environment (before OS/390 UNIX V2R9) with no regard to sysplex.

BPXPRMxx

```
FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
.
.
.
```

Figure 2. BPXPRMxx for Single System before OS/390 UNIX V2R9 or Later Environments

Note: The root can be mounted either read-only or read/write.

Figure 3 shows the recommended setup of the root HFS in a single system environment.

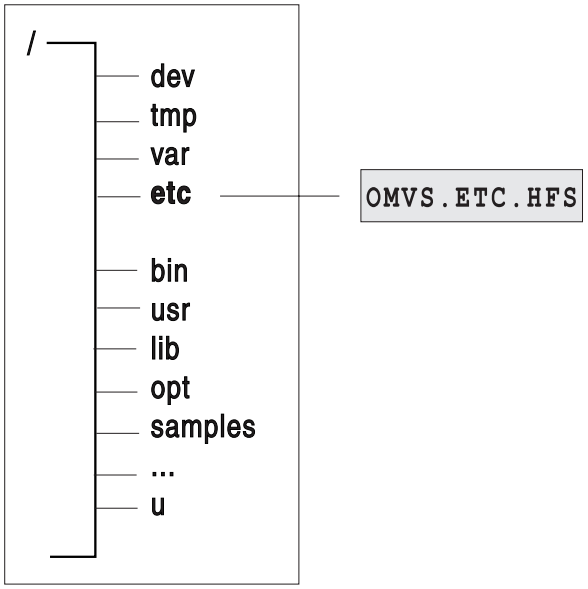

Figure 3. Single System before OS/390 UNIX V2R9

The directories in the root HFS represent “first-level” directories created by IBM. The **/etc**, **/dev**, **/var**, **/tmp**, and **/u** directories are used as mount points for other HFS data sets.

File Systems in Single System OS/390 UNIX V2R9 or Later Environments

Figure 4 shows what BPXPRMxx file system parameters would look like in an OS/390 UNIX V2R9 (or later) single system environment, and Figure 5 on page 6 shows the corresponding single system image. SYSPLEX(NO) is specified (or the default taken), and the mount mode is read/write.

Note: The root can be mounted either read-only or read/write.

BPXPRMxx

```
FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

SYSPLEX(NO)

ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.DEV.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/dev')

MOUNT
FILESYSTEM('OMVS.TMP.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/tmp')

MOUNT
FILESYSTEM('OMVS.VAR.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/var')

MOUNT
FILESYSTEM('OMVS.ETC.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/etc')
```

Figure 4. BPXPRMxx Parmlib Member for Single System: OS/390 UNIX V2R9

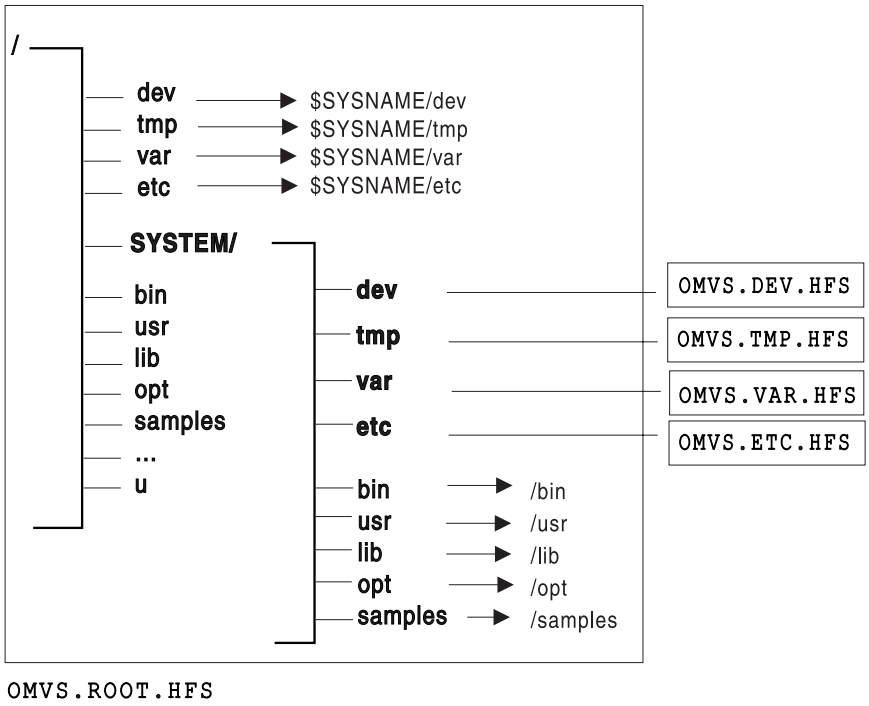

Figure 5. Single System: OS/390 UNIX V2R9

The presence of symbolic links is transparent to the user. In the illustrations used throughout this chapter, symbolic links are indicated with an arrow.

In Figure 5, the root file system contains an additional directory, **/SYSTEM**; existing directories, **/etc, /dev, /tmp** and **/var** are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment.

Note that if the content of the symbolic link begins with $SYSNAME and SYSPLEX is specified NO, then $SYSNAME is replaced with /SYSTEM when the symbolic link is resolved.

File Systems in OS/390 UNIX V2R9 or Later Sysplex Environments

This section describes file systems in sysplex environments (OS/390 UNIX V2R9 or later) and what you need to do to take advantage of shared HFS support, such as creating specific HFS data sets (also see Table 1 on page 3) and the OMVS Couple Data Set, updating COUPLExx, and customizing BPXPRMxx.

You must not assume that with shared HFS, two systems can share a common HFS data set for **/etc**, **/tmp**, **/var**, and **/dev**. This is not the case. Even with shared HFS, each system must have specific HFS data sets for each of these file systems. The file systems are then mounted under the system-specific HFS (see Figure 15 on page 20). With shared HFS support, one system can access system-specific file systems on another system. (Note that the existing security model remains the same.) For example, while logged onto SY2, you can gain read/write access to SY1's **/tmp** by specifying **/SY1/tmp/**.

You should also be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in shared HFS. The easiest way to accomplish this is to create a single

BPXPRMxx member that contains file system information for each system participating in shared HFS. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system. To see the differences between having one BPXPRMxx member for all participating systems and having one member for each participating system, see the two examples in “Scenario 2: Multiple Systems in the Sysplex – Using the Same Release Level” on page 18.

In addition, facilities required for a particular file system must be initiated on all systems in the participating group. For example, NFS requires TCP/IP; if you specify a filesystype of NFS, you must also initialize TCP/IP when you initialize NFS, even if there is no network connection.

Procedures for Establishing Shared HFS in a Sysplex

Creating the Sysplex Root HFS Data Set

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read/write and designated AUTOMOVE (see “Customizing BPXPRMxx for Shared HFS” on page 13 for a description of the AUTOMOVE BPXPRMxx parameter). Only one sysplex root is allowed for all systems participating in shared HFS. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB. After the job runs, the sysplex root file system structure would look like Figure 6:

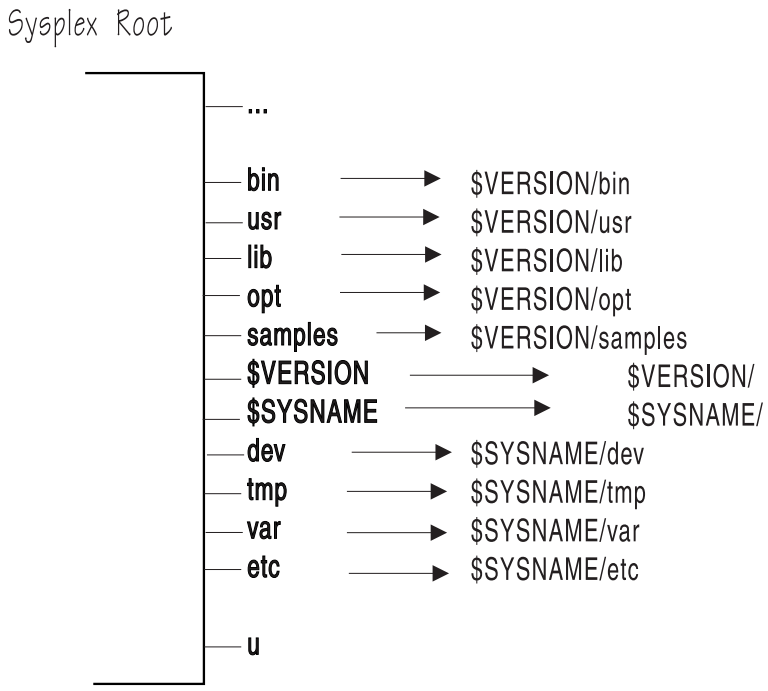

Figure 6. Sysplex Root

No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the

sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequent as possible.

Creating the System-Specific HFS Data Sets

Directories in the system-specific HFS data set are used as mount points, specifically for **/etc**, **/var**, **/tmp**, and **/dev**. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in shared HFS). After you invoke the job, the system-specific file system structure would look like Figure 7:

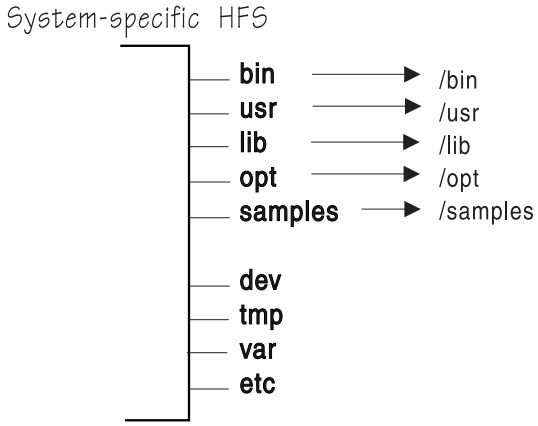

Figure 7. System HFS

The system-specific HFS data set should be mounted read/write, and should be designated NOAUTOMOVE (see “Customizing BPXPRMxx for Shared HFS” on page 13 for a description of the NOAUTOMOVE BPXPRMxx parameter). **/etc**, **/var**, **/tmp**, and **/dev** should also be mounted NOAUTOMOVE. I

n addition, IBM recommends that the name of the system-specific data set contain the system name as one of the qualifiers. This allows you to use the &SYSNAME symbolic (defined in IEASYMxx) in BPXPRMxx. “Shared HFS Implications during a Planned Shutdown of z/OS UNIX” on page 27.

Mounting the Version HFS

The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, “root HFS” has been renamed to “version HFS”.

Figure 8 on page 9 shows a version HFS.

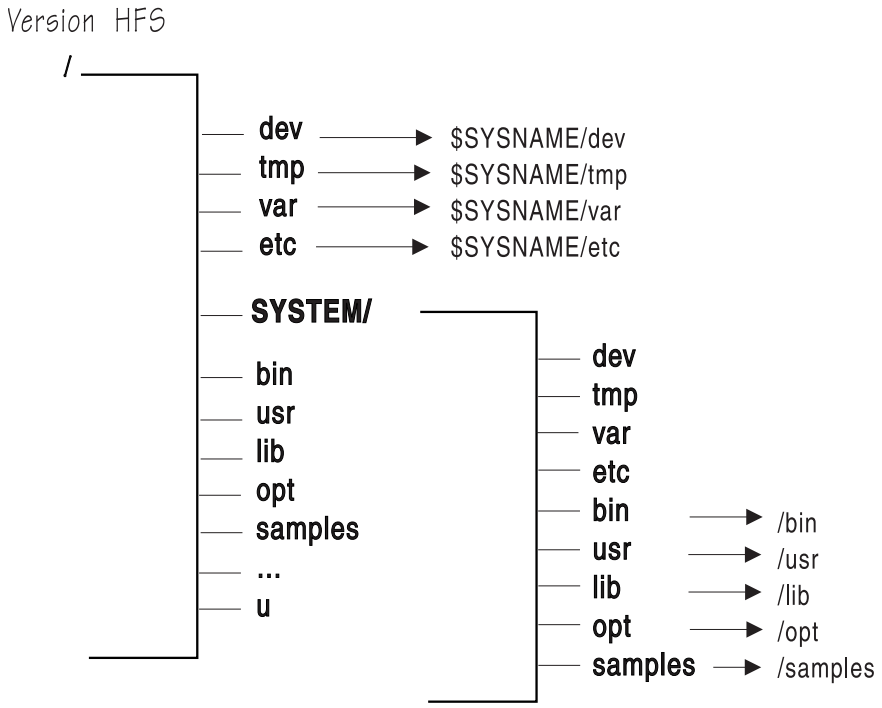

Figure 8. Version HFS

IBM recommends that you mount the version HFS read-only in a sysplex environment, and that you designate it AUTOMOVE. The mount point for the version HFS is dynamically created if the VERSION statement is used in BPXPRMxx.

Note: IBM does not recommend using &SYSNAME as one of the qualifiers for the version HFS data set name. In "Sysplex Scenarios Showing Shared HFS Capability" on page 15, REL9 and REL9A are used as qualifiers, which correspond to the system release levels. However, you do not necessarily have to use the same qualifiers. Other appropriate names are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer.

IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one version HFS for each set of systems participating in shared HFS and who are at the same release level (that is, using the same SYSRES volume). In other words, each version HFS denotes a different level of the system or a different service level. For example, if you have 20 systems participating in shared HFS and 10 of those systems are at OS/390 UNIX V2R9 and the the other 10 are at z/OS UNIX V1R1, then you'll have one version HFS for the OS/390 V2R9 systems and one for the z/OS UNIX V1R1 systems. In essence, you will have as many version HFSs for the participating systems as you have different levels running.

Before you mount your version HFS read-only, you may have some element-specific actions. These are described in "Mounting the Root HFS in Read-Only Mode" in the Managing the Hierarchical File System chapter.

Creating an OMVS Couple Data Set (CDS)

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the **primary** and the other is a backup that is referred to as the **alternate**. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

Use of the CDS functions in the following manner:

1. The first system that enters the sysplex with SYSPLEX(YES) initializes an OMVS CDS. The CDS controls shared HFS mounts and will eventually contain information about all systems participating in shared HFS.

   This system processes its BPXPRMxx parmlib member, including all its ROOT and MOUNT statement information. It is also the designated owner of the byte range lock manager for the participating group. The MOUNT and ROOT information are logged in the CDS so that other systems that eventually join the participating group can read data about systems that are already using shared HFS.
2. Subsequent systems joining the participating group will read what is already logged in the CDS and will perform all mounts. Any new BPXPRMxx mounts are processed and logged into the CDS. Systems already in the participating group will then process the new mounts added to the CDS.

Figure 9 on page 11 shows the sample JCL with comments. The statements in bold contain the values you specify based on your environment.

```
//*
//STEP10    EXEC PGM=IXCL1DSU
//STEPLIB   DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
  /* Begin definition for OMVS couple data set(1)            */
     DEFINEDS SYSPLEX(PLEX1)       /* Name of the sysplex in
                                      which the OMVS couple data
                                      set is to be used.      */
              DSN(SYS1.OMVS.CDS01) VOLSER(3390x1)
                                    /* The name and
                                       volume for the OMVS
                                       couple data set.  The
                                       utility will allocate a
                                       new data set by the name
                                       specified on the volume
                                       specified.              */
             MAXSYSTEMS(8)
                                    /* Specifies the number
                                       of systems to be supported
                                       by the OMVS CDS.
                                       Default =     8          */

            NOCATALOG             /* Default is not to CATALOG */
           DATA TYPE(BPXMCDS)       /* The type of data in the
                                       data set being created is
                                       for OMVS.  BPXMCDS is the
                                       TYPE for OMVS.          */
              ITEM NAME(MOUNTS) NUMBER(500)
                                    /* Specifies the number of
                                       MOUNTS that can be supported
                                       by OMVS.
                                       Default =   100
                                       Minimum =     1
                                       Maximum = 50000          */
             ITEM NAME(AMTRULES) NUMBER(50)
                                    /* Specifies the number
                                       of automount rules that can
                                       be supported by OMVS.
                                       Default =    50
                                       Minimum =    50
                                       Maximum =  1000           */
```

Figure 9. BPXISCDS: Sample z/OS JCL for Formatting a Couple Data Set (Part 1 of 2)

```
/* Begin definition for OMVS couple data set(2)              */
   DEFINEDS SYSPLEX(PLEX1)          /* Name of the sysplex in
                                       which the OMVS couple data
                                       set is to be used.      */
            DSN(SYS1.OMVS.CDS02) VOLSER(3390x2) /* The name and
                                       volume for the OMVS
                                       couple data set.  The
                                       utility will allocate a
                                       new data set by the name
                                       specified on the volume
                                       specified.               */
            MAXSYSTEMS(8)
                                    /* Specifies the number
                                       of systems to be supported
                                       by the OMVS CDS.
                                       Default =     8          */

            NOCATALOG              /* Default is not to CATALOG */
          DATA TYPE(BPXMCDS)        /* The type of data in the
                                       data set being created is
                                       for OMVS.  BPXMCDS is the
                                       TYPE for OMVS.           */
              ITEM NAME(MOUNTS) NUMBER(500)
                                    /* Specifies the number of
                                       MOUNTS that can be supported
                                       by OMVS.
                                       Default =   100
                                       Minimum =    10
                                       Maximum = 50000           */
              ITEM NAME(AMTRULES) NUMBER(50)
                                    /* Specifies the number
                                       of automount rules that can
                                       be supported by OMVS.
                                       Default =     50
                                       Minimum =     50
                                       Maximum =   1000          */
```

Figure 9. BPXISCDS: Sample z/OS JCL for Formatting a Couple Data Set (Part 2 of 2)

Note: Automount mounts must be included in the MOUNTS value. The number of automount mounts is the expected number of concurrently mounted file systems using the automount facility. For example, you may have specified 1000 file systems to be automounted, but if you expect only 50 to be used concurrently, you should factor these 50 into your MOUNTS value.

For more information about setting up a sysplex on MVS and descriptions of XCF and CDS, see *z/OS MVS Setting Up a Sysplex*.

The NUMBER(*nnnn*) specified for mounts and automount rules (a generic or specific entry in an automount map file) is directly linked to function performance and the size of the CDS. If maximum values are specified, the size of the CDS will increase accordingly and the performance level for reading and updating it will decline.

Conversely, if the NUMBER values are too small, the function (for example, the number of mounts supported) would fail after the limit is reached. However, a new CDS can be formatted and switched in with larger values specified in NUMBER. To make the switch, issue the SETXCF COUPLE,PSWITCH command. For more information on this command, see “Considerations for all Couple Data Sets” in *z/OS*

MVS Setting Up a Sysplex. The number of file systems required (factoring in an additional number to account for extra mounts), determines your minimum and maximum NUMBER value.

After the CDS is created, it must be identified to XCF for use by z/OS UNIX.

Updating COUPLExx to Define the OMVS CDS to XCF

Update the active COUPLExx parmlib member to define a primary and alternate OMVS CDS to XCF. The primary and alternate CDSs should be placed on separate volumes. (Figure 9 on page 11 shows the primary CDS on volume 3390x1 and the secondary CDS on 3390x2.)

Figure 10 shows the COUPLExx parmlib member; statements that define the CDS are in bold.

```
/* For all systems in any combination, up to an eightway  */
COUPLE INTERVAL(60)                    /* 1 minute */
   OPNOTIFY(60)                        /* 1 minute */
   SYSPLEX(PLEX1)                      /* SYSPLEX NAME*/
   PCOUPLE(SYS1.PCOUPLE,CPLPKP)        /* COUPLE DS */
   ACOUPLE(SYS1.ACOUPLE,CPLPKA)        /* ALTERNATE DS*/
   MAXMSG(750)
   RETRY(10)
DATA TYPE(CFRM)
   PCOUPLE(SYS1.PFUNCT.CTTEST,FDSPKP)
   ACOUPLE(SYS1.AFUNCT.CTTEST,FDSPKA)
DATA TYPE(BPXMCDS)
   PCOUPLE(SYS1.OMVS.CDS01,3390x1)
   ACOUPLE(SYS1.OMVS.CDS02,3390x2)
/* CTC DEFINITIONS: ALL SYSTEMS  */
PATHOUT DEVICE(8E0)
PATHIN DEVICE(CEF)
```

Figure 10. COUPLExx Parmlib Member

The MVS operator commands (DISPLAY XCF, SETXCF, DUMP, CONFIG, and VARY) enable the operator to manage the z/OS UNIX CDS. For a complete description of these commands, see *z/OS MVS System Commands*.

Customizing BPXPRMxx for Shared HFS

HFS sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbolics. Figure 13 on page 18 shows an example of this unified member. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. An example of this is Figure 14 on page 19.

The following parameters set up HFS sharing in a sysplex:

- SYSPLEX(YES) sets up HFS sharing for those who are in the SYSBPX XCF group, the group that is participating in HFS data sharing. To participate in HFS data sharing, the systems must be at the OS/390 V2R9 level or later. Those system that specify SYSPLEX(YES) make up the participating group for the sysplex.

  If SYSPLEX(YES) is specified in the BPXPRMxx member, but the system is initialized in XCF-local mode, either by specifying COUPLE SYSPLEX(LOCAL) in the COUPLExx member or by specifying PLEXCFG=XCFLOCAL in the

IEASYSxx member, then the kernel will ignore the SYSPLEX(YES) value and initialize with SYSPLEX(NO). This system will not participate in shared HFS support after the initialization completes.

- VERSION('nnnn') allows multiple releases and service levels of the binaries to coexist and participate in HFS sharing. *nnnn* is a qualifier to represent a level of the version HFS. The most appropriate values for *nnnn* are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer. A directory with the value *nnnn* specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version HFS.

  There is one version HFS for every instance of the VERSION parameter. More information about version HFS appears in "Mounting the Version HFS" on page 8.
- The SYSNAME(sysname) parameter on ROOT and MOUNT statements specifies the particular system on which a mount should be performed. *sysname* is a 1–8 alphanumeric name of the system. This system will then become the owner of the file system mounted. The owning system must be IPLed with SYSPLEX(YES). IBM recommends that you specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

  The SYSNAME parameter is also used with SETOMVS when moving file systems, as demonstrated in "Moving File Systems in a Sysplex" on page 24
- The AUTOMOVE|NOAUTOMOVE parameters on ROOT and MOUNT indicate what happens to the file system if the system that owns that file system goes down. AUTOMOVE specifies that ownership of the file system automatically moves to another system. NOAUTOMOVE specifies that the file system will not be moved if the owning system goes down and the file system is inaccessible. AUTOMOVE is the default.

  You should define your version and sysplex root HFS data sets as AUTOMOVE, and define your system-specific HFS data sets as NOAUTOMOVE. Do not define a file system as NOAUTOMOVE and a file system underneath it as AUTOMOVE; in this case, the file system defined as AUTOMOVE will not be recovered after a system failure until that failing system is restarted.

  Note: To ensure that the root is always available, use the default: AUTOMOVE.

  For file systems that are mostly used by DFS clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will not change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS. Specifying NOAUTOMOVE is recommended because a file system can only be exported by the DFS server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported from DFS. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS clients. When an owning system recycles and a DFS-exported file system has been taken over by one of the other systems, DFS cannot automatically re-export that file system. The file system will have to be moved from its current owner back to the original DFS system, the one that has just been recycled, and then exported again.

If file systems are mounted read-only on all systems, the owner is the first system, with connectivity to the DASD, that processes the mount. If that system is taken down and other systems have the file system mounted read-only with connectivity to the DASD, the ownership will change to one of those systems, no matter what the value of the AUTOMOVE statement is.

For more information about VERSION, SYSPLEX, SYSNAME and AUTOMOVE|NOAUTOMOVE, see *z/OS MVS Initialization and Tuning Reference*.

Sysplex Scenarios Showing Shared HFS Capability

Scenario 1: First System in the Sysplex

Figure 11 and Figure 12 on page 17 shows a z/OS UNIX file system configuration for shared HFS. Here, SYSPLEX(YES) and a value on VERSION are specified, and a directory is dynamically created on which the version HFS data set is mounted. This type of configuration requires a sysplex root and system-specific HFS.

```
BPXPRMxx for (SY1)

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')          1
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')   2
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')               3
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')          4
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')          5
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 11. BPXPRMxx Parmlib Setup — HFS Sharing

1 This is the sysplex root HFS data set, and was created by running the BPXISYSR job. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

2 This is the system-specific HFS data set, and was created by running the BPXISYSS job. It must be mounted read/write. NOAUTOMOVE is specified

because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME.** will resolve to **/SY1** during parmlib processing. This mount point is created dynamically at system initialization.

3 This is the old root HFS (version HFS). IBM recommends that it be mounted read-only. Its mount point is created dynamically and the name of the HFS is the value specified on the VERSION statement in the BPXPRMxx member. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

4 This HFS contains the system-specific **/dev** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./dev** will resolve to **/SY1/dev** during parmlib processing.

5 This HFS contains system-specific **/tmp** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move to another system should the owning system go down. The MOUNTPOINT statement **/&SYSNAME./tmp** will resolve to **/SY1/tmp** during parmlib processing.

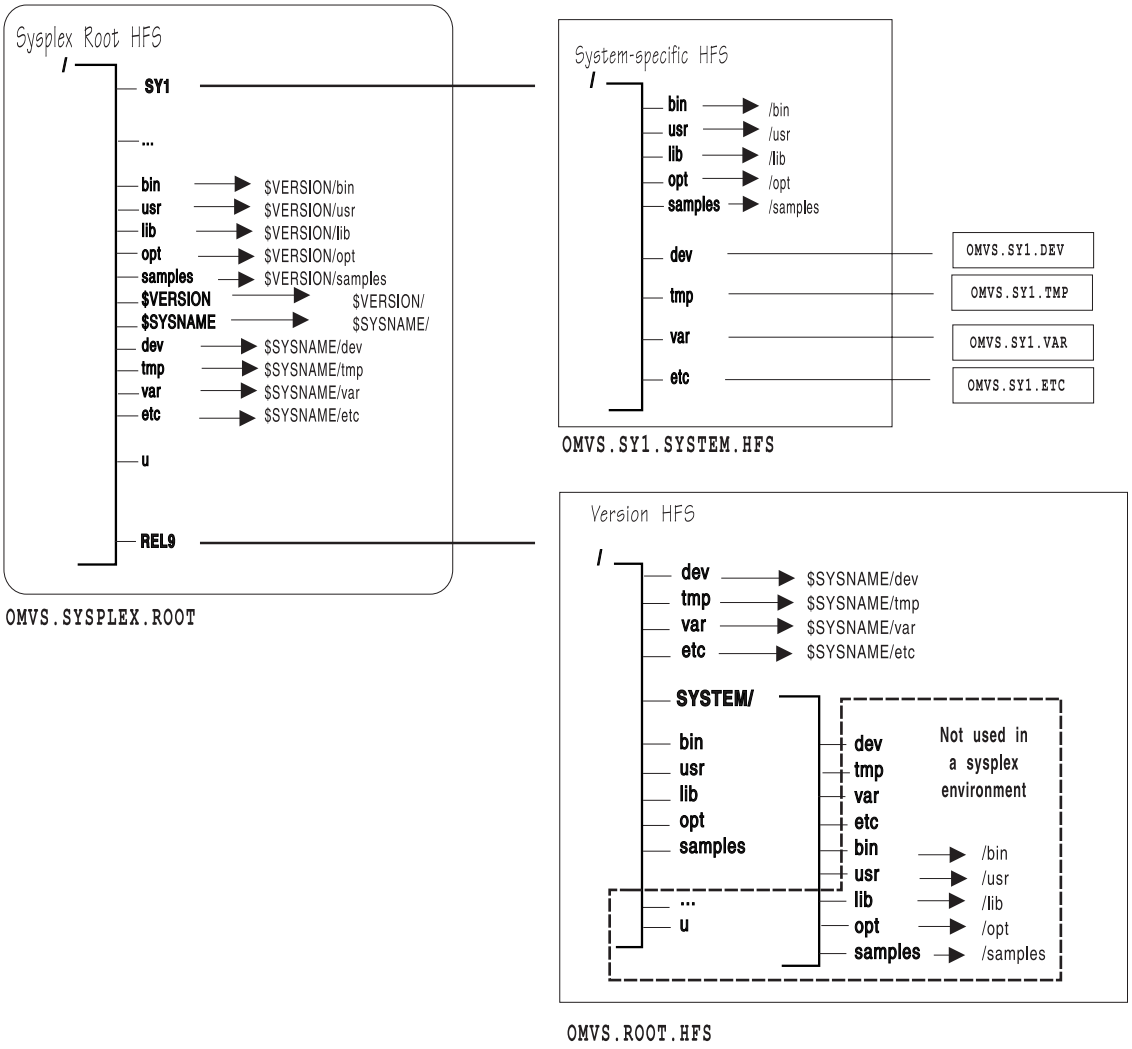

Figure 12. HFS Sharing in a Sysplex

If the content of the symbolic link begins with $VERSION or $SYSNAME, the symbolic link will resolve in the following manner:

- If you have specified SYSPLEX(YES) and the symbolic link for **/dev** has the contents **$SYSNAME/dev**, the symbolic link resolves to **/SY1/dev** on system SY1 and **/SY2/dev** on system SY2.
- If you have specified SYSPLEX(YES) and the content of the symbolic link begins with $VERSION, $VERSION resolves to the value *nnnn* specified on the VERSION parameter. Thus, if VERSION in parmlib is set to REL9, then $VERSION resolves to /REL9. For example, a symbolic link for **/bin**, which has the contents **$VERSION/bin**, resolves to **/REL9/bin** on a system whose $VERSION value is set to REL9.

In the above scenario, if **ls –l /bin/** is issued, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the pathname **/REL9/bin**. The contents of **/REL9/bin** will now be displayed.

Scenario 2: Multiple Systems in the Sysplex – Using the Same Release Level

Figure 15 on page 20 shows another SYSPLEX(YES) configuration. In this configuration, however, two or more systems are sharing the same version HFS (the same release level of code). Figure 13 shows a sample BPXPRMxx for the entire sysplex (what IBM recommends) using &SYSNAME. as a symbolic name, and Figure 14 on page 19 shows a configuration where each system in the sysplex has its own BPXPRMxx. For our example, SY1 has its own BPXPRMxx and SY2 has its own BPXPRMxx.

One BPXPRMxx Member to Define File Systems for the Entire Sysplex

```
FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 13. Sharing HFS Data Sets: One Version HFS and One BPXPRMxx for Entire Sysplex

```
BPXPRMS1 (for SY1)                      BPXPRMS2 (for SY2)

FILESYSTYPE                             FILESYSTYPE
TYPE(HFS)                               TYPE(HFS)
ENTRYPOINT(GFUAINIT)                    ENTRYPOINT(GFUAINIT)
PARM(' ')                               PARM(' ')

VERSION('REL9')                         VERSION('REL9')
SYSPLEX(YES)                            SYSPLEX(YES)

ROOT                                    ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')         FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                    TYPE(HFS) MODE(RDWR)

MOUNT                                   MOUNT
FILESYSTEM('OMVS.SY1.SYSTEM.HFS')       FILESYSTEM('OMVS.SY2.SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1')                      MOUNTPOINT('/SY2')

MOUNT FILESYSTEM('OMVS.ROOT.HFS')       MOUNT FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)                    TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                 MOUNTPOINT('/$VERSION')

MOUNT FILESYSTEM('OMVS.SY1.DEV')        MOUNT FILESYSTEM('OMVS.SY2.DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/dev')                  MOUNTPOINT('/SY2/dev')

MOUNT FILESYSTEM('OMVS.SY1.TMP')        MOUNT FILESYSTEM('OMVS.SY2.TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE         TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/SY1/tmp')                  MOUNTPOINT('/SY2/tmp')
.
.
.
```

Figure 14. Sharing HFS Data Sets: One Version HFS and Separate BPXPRMxx Members for Each System in the Sysplex

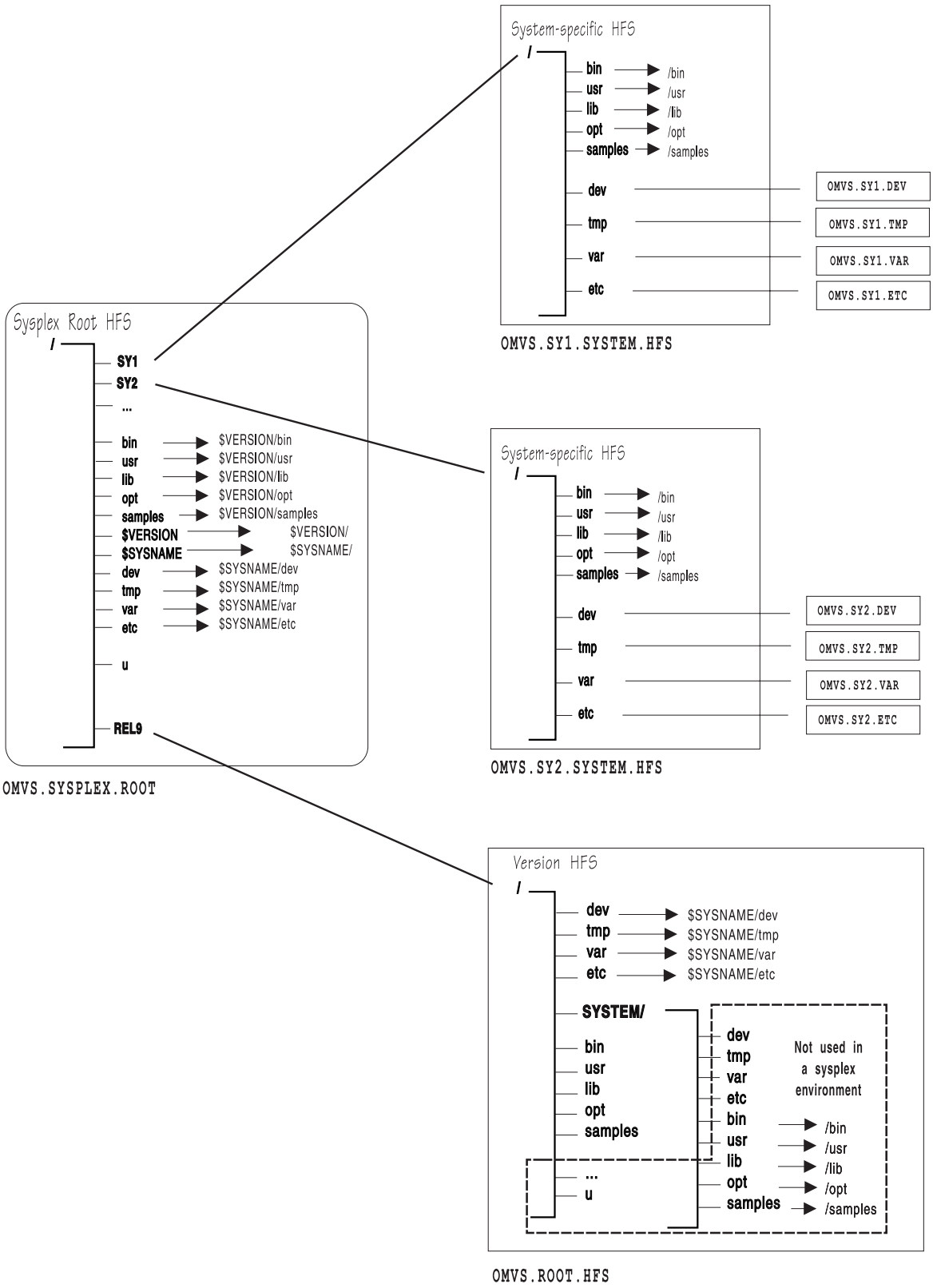

Figure 15. Sharing HFS Data Sets in a Sysplex for Release 9– Multiple Systems in a Sysplex Using the Same Release Level

In this scenario, where multiple systems in the sysplex are using the same version HFS, if **ls –l /bin/** is issued from either system, the user expects to see the contents

of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to **/REL9** which makes the pathname **/REL9/bin**. The contents of this directory will be displayed.

Scenario 3: Multiple Systems in a Sysplex Using Different Release Levels

If your participating group is in a sysplex that runs multiple levels of z/OS and/or OS/390, your configuration might look like the one in Figure 17 on page 22. In that configuration, each system is running a different level of z/OS and, therefore, has different version HFS data sets; SY1 has the version HFS named OMVS.SYSR9A.ROOT.HFS and SY2 has the version HFS named OMVS.SYSR9.ROOT.HFS. Figure 16 shows two BPXPRMxx parmlib members that define the file systems in this configuration. Figure 18 on page 23 shows a single BPXPRMxx parmlib member that can be used to define this same configuration; it uses &SYSR1. as the symbolic name for the two version HFS data sets.

```
BPXPRMxx (for SY1)                          BPXPRMxx (for SY2)

FILESYSTYPE                                 FILESYSTYPE
TYPE(HFS)                                   TYPE(HFS)
ENTRYPOINT(GFUAINIT)                        ENTRYPOINT(GFUAINIT)
PARM(' ')                                   PARM(' ')

VERSION('REL9A')                            VERSION('REL9')
SYSPLEX(YES)                                SYSPLEX(YES)

ROOT                                        ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')             FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)                        TYPE(HFS) MODE(RDWR)

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')     FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')                    MOUNTPOINT('/&SYSNAME.')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.SYSR9A.ROOT.HFS')          FILESYSTEM('OMVS.SYSR9.ROOT.HFS')
TYPE(HFS) MODE(READ)                        TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')                     MOUNTPOINT('/$VERSION')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')            FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')                MOUNTPOINT('/&SYSNAME./dev')

MOUNT                                       MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')            FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE             TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')                MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 16. BPXPRMxx Parmlib Setup for Multiple Systems Sharing HFS Data Sets and Using Different Release Levels

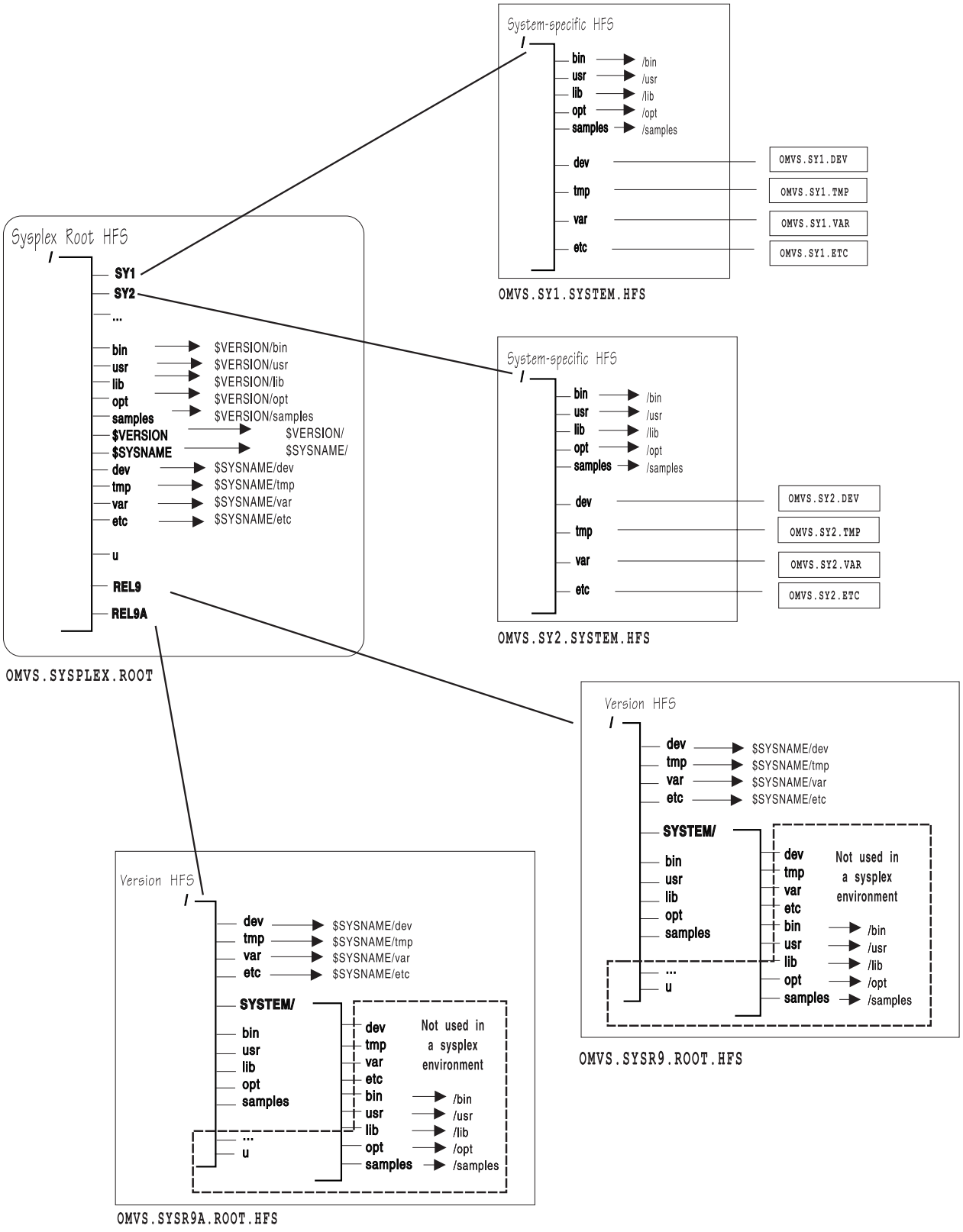

Figure 17. Sharing HFS Data Sets between Multiple Systems Using Different Release Levels

In this scenario, for example, if **ls –l /bin/** is issued on SY1, the user expects to see the contents of **/bin**. However, because **/bin** is a symbolic link pointing to **$VERSION/bin**, the symbolic link must be resolved first. $VERSION resolves to

/SYSR9A on SY1, which makes the pathname **/SYSR9A/bin**. The contents of this directory will now display. If **ls –l /bin/** is issued on SY2, the contents of **/SYSR9/bin** will display.

From SY2 you can display information on SY1 by fully qualifying the directory. For example, to view SY1's **/bin** directory, you issue **ls –l /SY1/bin/**.

```
One BPXPRMxx Member to Define File Systems for the Entire Sysplex
Using Different Releases

FILESYSTYPE
TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')

VERSION('&SYSR1.')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.&SYSR1..ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 18. One BPXPRMxx Parmlib Member for Multiple Systems Sharing HFS Data Sets and Using Different Release Levels

In order to use one BPXPRMxx parmlib file system member, we have used another system symbolic like &SYSR1. This system symbolic is used in the VERSION parameter and also as a qualifier in the version HFS data set name.

Keeping Automount Policies Consistent on All Systems in the Sysplex

IBM recommends that you keep the automount policies consistent across all the participating systems in the sysplex.

Before OS/390 UNIX V2R9, your automount policy most likely resided in the **/etc/auto.master** and **/etc/u.map files**. For those using shared HFS, each participating system has a separate **/etc** file system. In order for automount policy to be consistent across participating systems, the same copy of the automount policy must exist in every system's **/etc/auto.master** and **/etc/u.map** files.

For example both SY1 and SY2 have the following files:

- **/etc/auto.master**

```
/u     /etc/u.map
```

- **/etc/u.map**

```
name       *
type       HFS
filesystem OMVS.<uc_name>.HFS
mode       rdwr
duration   60
delay      60
```

When the automount daemon initializes on SY1, it will read its local **/etc/auto.master** file to identify what directories to manage; in this case, it is **/u**. Next, the automount daemon will use the policy specified in the local **/etc/u.map** file to mount file systems with the specified naming convention under **/u**. The automount daemon on SY2 will perform similar actions. Because all mounted file systems are available to all participating systems in the sysplex, your automount policy must be consistent. This is true for the file system name specified in **/etc/u.map** and the values for other parameters in **/etc/u.map** and **/etc/auto.master**.

Moving File Systems in a Sysplex

You may need to change ownership of the file system for recovery or re-IPLing. To check for file systems that have already been mounted, use the **df** command from the shell.

The SETOMVS command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex, or you can use the **chmount** command from the shell. However, do not move two types of file systems:

- System-specific file systems
- File systems that are being exported by DFS. You have to unexport them from DFS first and then move them

Examples of moving file systems are:

1. To move ownership of the file system that contains **/u/wjs** to SY1:

   ```
   chmount -d SY1 /u/wjs
   ```

2. To move ownership of the payroll file system from the current owner to SY2 using SETOMVS, issue:

   ```
   SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SY2
   ```

   or (assuming the mount point is over directory **/PAYROLL**)

   ```
   SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SY2
   ```

Shared HFS Implications During System Failures and Recovery

File system recovery in a shared HFS environment takes into consideration file system specifications such as AUTOMOVE, NOAUTOMOVE, and whether or not the file system is mounted read-only or read/write.

Generally, when an owning system fails, ownership over its automove-mounted file system is moved to another system and the file is usable. However, if a file system is mounted read/write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered. (The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.)

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned; if this happens, you will see message BPXF213E. This is true if the file system is mounted either read/write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that are owned by another system are still usable. However, all file operations for the unowned file system will fail until a new owner is established. The shared HFS support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared HFS. Should a subsequent recovery attempt succeed, the file system transitions from the unowned to the active state.

Applications using files in unowned file systems will need to close (BPX1CLO) those files and reopen (BPX1OPN) them after the file system is recovered.

File systems that are mounted NOAUTOMOVE will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Note that since the file system still exists in the file system hierarchy, the file system mount point is still in use.

| File systems that are mounted NOAUTOMOVE will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Note that since the file system still exists in the file system hierarchy, the file system mount point is still in use. File systems that are mounted below a NOAUTOMOVE file system will not be accessible via pathname when the NOAUTOMOVE file system becomes unavailable.

| It is not recommended that you mount AUTOMOVE file systems within NOAUTOMOVE file systems. When a NOAUTOMOVE file system becomes unowned and there are AUTOMOVE file systems mounted within it, those AUTOMOVE file systems will retain a level of availability, but only for files that are already open. When the NOAUTOMVE file system becomes unowned, it will not be possible to perform pathname lookup through it to the file systems mounted within

it, making those file systems unavailable for new access. When ownership is restored to the unowned file system, access to the file systems mounted within it will also be restored.

Movement of data

File systems can be managed so as to maximize their availability when systems exit the participating group. You have more control over this when the outage is planned, but there are steps you can take to help manage the placement of data in the event of a system failure.

Recovery processing for the file systems that are owned by a failed system is managed internally by all the systems in the participating group. If you want special considerations for the placement of certain file systems, you can use the options provided by the various mount services to specify the original owner and subsequent owners for a particular file system.

To assist with file system management, the MOUNT command supports the following automove options:

- SYSNAME() indicates teh initial owner of the file system.
- AUTOMOVE options:
  - NOAUTOMOVE indicates that no attempt will be made to keep the file system active when the current owner fails. The file system will remain in the hierarchy for possible recovery when the original owner reinitializes. Use this option on mounts for system-specific file systems if you want to have automatic recovery when the original owner rejoins the participating group.

    When the NOAUTOMOVE option is used, the file system becomes unowned when the owning system exits the participating group. The file system remains unowned until the last owning system restarts, or until the file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use.

    An unowned file system is a mounted file system that does not have an owner. Because it still exists in the file system hierarchy, it can be recovered or unmounted.
  - AUTOMOVE indicates that recovery of the file system is to be performed to a randomly selected owner when the current owner fails. This option is recommended for use on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.
- Do not mount automoveable file systems within a fiel system that is mounted NOAUTOMOVE.

Note: To ensure that the root is always available use the default: AUTOMOVE.

For file systems that are mostly used by DFS clients, consider specifying NOAUTOMOVE on the MOUNT statement. Then the file systems will now change ownership if the system is suddenly recycled, and they will be available for automatic re-export by DFS. Specifying NOAUTOMOVE is recommended because a file system can only be exported by the DFS server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported from DFS. When recovering from system outages, you need to weigh sysplex availability against availability to the DFS clients. When an owning system recycles and a DFS-exported file system has been taken over by one of the other systems, DFS cannot automatically re-export that file system. The file system will have to be moved from its current owner back to the original DFS system, the one that has just been recycled, and then exported again.

Most of the z/OS UNIX interfaces that provide for mounting file systems (such as TSO, shell, ISHELL, and BPX2MNT) support some form of the options described here. See the associated documentation for the exact syntax.

Shared HFS Implications during a Planned Shutdown of z/OS UNIX

"File System Shutdown" on page 38 describes the procedures to use when you are planning a shutdown of z/OS UNIX. It is important that you understand the system actions that result when you use those procedures.

The current automove option dictates if and how the participating group recovers file system ownership from an exited system. It has no effect on the manual movement of the file system. However, when you are using the procedures for shutting down z/OS UNIX to prepare for a planned system outage, the automove option does apply. This can be explained with the following rationale:

- A system failure does not provide any means for manual intervention. The automove option provides a set of rules for automatic recovery.
- A request to move a file system manually is a deliberate action on behalf of an authorized user or administrator, and should override any rules for automatic recovery
- Using tools to prepare for a system outage is also a deliberate action on behalf of an authorized user or administrator, but you are using these tools in an environment that can be customized to allow for additional manual intervention. You can synchronize data before the system outage, and then manage the planned outage in the same way as the unplanned outage, by making use of the automatic recovery rules that are supplied by the automove options. If you prefer some other action, you can perform manual intervention to move specific file system ownership before you use these methods for shutdown preparation.

  There are two system commands that prepare file systems for a system shutdown:

  – F BPXOINIT,SHUTDOWN=FILESYS
  – F BPXOINIT,SHUTDOWN=FILEOWNER

  The F BPXOINIT,SHUTDOWN=FILESYS system command removes file system ownership from the system in preparation for a system shutdown. File systems are moved according to the automove options or unmounted. Shared HFS activity on the active systems in the participating group can still have an effect on the system on which F BPXOINIT,SHUTDOWN=FILESYS has been issued. To prevent the issuing system from becoming the new owner of file systems that are being relocated by another system, shutdown processing should be coordinated with other systems. Entering the SHUTDOWN=FILESYS command does not prevent the system from being eligible to own any file system that is being mounted, moved, or recovered, including the ones that were just moved off the system. Once SHUTDOWN=FILESYS has completed processing, the system is free to take ownership of file systems as a result of activity on this system or another system.

  The F BPXOINIT,SHUTDOWN=FILEOWNER system command removes file system ownership from the system in preparation for a system shutdown. File systems are moved according to the automove options or unmounted. The system is disabled from becoming a file system owner via move or recovery options until z/OS UNIX has been recycled. However, new mounts (where this system is the file owner) are not blocked. Avoid mounting file systems (where this system is the target owner) during shutdown operations. **When using this**

command, you do not need to coordinate shutdown processing with other systems in the shared HFS configuration.

Note: Automounted file systems are always AUTOMOVE on V1R2 and lower. Failures during this processing cause message BPXM054I to be displayed, and file systems may remain owned by the system when shutdown completes. See “Shared HFS Implications during a Planned Shutdown of z/OS UNIX” on page 27 for more information.

When a system is removed from the sysplex, there is a window of time during which any file systems it owned will become inaccessible to other systems. This window occurs while other systems are being notified of the system's exit from the sysplex, and before they start the cleanup for that system. Ideally, ownership of critical file systems will have been moved to other systems before the system exits, but if it has not, there will be a period of time during which these critical file system are unowned. It is then possible for another system that is performing initialization to see mount failures because critical file systems cannot be accessed. These failures can cause the initializing system to come up with out the necessary file systems from being mounted. To avoid this situation, it is important to make sure that any system that is being removed from the sysplex does not own any critical file systems.

File System Initialization

When you are preparing to bring a system back into the participating group after it has left, it is helpful to understand the coordination that occurs among the systems that are already participating in the group. You might see delays in the availability of the entering system because of activity occurring elsewhere in the sysplex. Although it is possible to bring up multiple systems simultaneously, when they reach the point of z/OS UNIX initialization, their processing is serialized so as to allow only one system at a time to initialize z/OS UNIX.

Other examples of activities occurring on other active systems that can cause the initializing system to experience delays are

- Unmounting a file system
- Changing ownership of a file system
- Recovering for systems that have left the participating group

Before it rejoins the participating group, a system processes all the file systems that are listed in the current hierarchy of the participating group. It also attempts to reclaim any unowned file systems that it previously owned when it was part of the participating group. It does not attempt to reclaim those file systems that were successfully moved or recovered to another system in the sysplex.

During initialization, any new MOUNT statements in the BPXPRMxx parmlib member are processed, which makes those file systems available for use within the participating group after they are successfully mounted.

While a system is initializing in a sysplex, critical file systems that are necessary for initialization to complete succesfully might become unavailable due to a system outage. When a system is removed from the sysplex, there is a window of time during which any file systems it owned will become inaccessible to other systems. This window of time occurs while other systems are being notified of the system's exit from the sysplex and before they start the cleanup for that system.

Ideally, ownership of critical file systems will have been moved to other systems before the system exits. If that has not happened, there will be a window of time during which these critical file systems are unowned. If the initializing system requires access to these critical file systems during this window, there will likely be mount failures that prevents the initialization from completing successfully. To avoid this situation, you must make sure that any system that is being removed from the sysplex does not own any critical file systems.

Locking Files

Users can lock all or part of a file that they are accessing for read/write purposes by using the Byte Range Lock Manager (BRLM) within the system.

The lock manager is initialized on only one system in the sysplex. The first system that enters the sysplex initializes the lock manager and becomes the system that owns the manager. For example, if SY1 is the first system in the sysplex, then SY1 owns the BRLM; all lock requests are routed to SY1.

When a system failure occurs on the system owning the BRLM, all history of byte range locks is lost. A new BRLM is established by one of the surviving systems in the sysplex, and locking can begin once that recovery has completed. However, to maintain data integrity after a failure where byte range locks are lost, z/OS UNIX provides the following information to processes which have used byte range locking:

- Access to open files for which byte range locks are held by any process will result in an I/O error. The file must be closed and reopened before use can continue.
- A signal is issued to any process which has made use of byte range locking. By default, a SIGTERM signal is issued against every such process, and an EC6 abend with reason code 0D258038 will terminate the process. If you do not want the process to be terminated, the process can use BPX1PCT (the physical file system control callable service) to specify a different signal for z/OS UNIX to use for notifying the process that the BRLM has failed. Any signal can be used for this purpose, thus allowing the user or application the ability to catch or ignore the signal and react accordingly.

  The system completion code EC6 and its associated reason codes are described in *z/OS MVS System Codes*. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about BPX1PCT.

Preparing File Systems for Shutdown

Chapter 2, “Managing Operations”, on page 33 describes the recommended procedures for shutting down z/OS UNIX.

Mounting File Systems Using NFS Client Mounts

With the z/OS NFS server, the client has remote access to z/OS UNIX files from a client workstation. Using the Network File System, the client can mount all or part of the file system and make it appear as part of its local file system. From the workstation, the client user can create, delete, read, write, and treat the host-located files as part of the workstation’s own file system.

In a similar way, the z/OS NFS client gives users remote access to files on an NFS server. Using NFS, the user can mount all or part of the remote file system and

make it appear as part of the local z/OS file hierarchy. From there, the user can create, delete, read, write, and treat the remotely located files as part of the own file system.

In a sysplex, the NFS Client-NFS Server relationship is as follows: the data that becomes accessible is accessible from any place in the sysplex as long as at least one of the systems has connectivity to the NFS server.

Note: Entries in the NFS Server Export Data Set can control which HFS directories can be mounted by client users. When specifying path names in this data set, you must specify fully qualified path names. That is, the use of symbolic links in this data set are not supported.

Tuning z/OS UNIX Performance in a Sysplex

The intersystem communication required to provide the additional availability and recoverability associated with z/OS UNIX shared HFS support, affects response time and throughput on R/W file systems being shared in a sysplex.

For example, assume that a user on SY1 requests a read on a file system mounted R/W and owned by SY2. Using shared HFS support, SY1 sends a message requesting this read to SY2 via an XCF messaging function:

```
SY1 ===> (XCF messaging function) ===> SY2
```

After SY2 gets this message, it issues the read on behalf of SY1, and gathers the data from the file. It then returns the data via the same route the request message took:

```
SY2 ===> (XCF messaging function) ===> SY1
```

Thus, adding z/OS UNIX to a sysplex increases XCF message traffic. To control this traffic, closely monitor the number and size of message buffers and the number of message paths within the sysplex. It is likely that you will need to define additional XCF paths and increase the number of XCF message buffers above the minimum default. For more information on signaling services in a sysplex environment, see *z/OS MVS Setting Up a Sysplex*.

You should also be aware that because of I/O operations to the CDS, every mount request requires additional system overhead. Mount time increases as a function of the number of mounts, the number of members in a sysplex, and the size of the CDS. You will need to consider the effect on your recovery time if a large number of mounts are required on any system participating in shared HFS.

DFS Considerations

A file system can only be exported by the DFS server at the system that owns the file system. Once a file system has been exported by DFS, it cannot be moved until it has been unexported by DFS.

To recover from system outages, you need to weigh sysplex availability against availability to the DFS and Server Message Block (SMB) clients. When an owning system recycles and a DFS-exported file system has been taken over by one of the other systems, DFS will not be able to automatically reexport that file system. The file system will have to be moved from its current owner back to the original DFS system, the one that has just been recycled, and then reexported.

For file systems that are mostly for use by DFS clients, you should consider specifying NOAUTOMOVE on the MOUNT statement so that they will not be taken over if the system is recycled, and they will be available for automatic reexport by DFS.

Chapter 2. Managing Operations

z/OS UNIX is designed to be continually available. This chapter discusses these tasks, which are done by operators.

Task	Topic
Stopping Processes	33
Terminating Threads with the MODIFY Command	34
Shutting Down z/OS UNIX	35
Dynamically Changing the BPXPRMxx Parameter Values	38
Tracing Events in z/OS UNIX	43
Displaying the Status of the Kernel	44
Taking a Dump of the Kernel and User Processes	46
Recovering from a Failure	48
Managing Interprocess Communication (IPC)	50

For information about the CANCEL, DISPLAY, MODIFY MSGRT, and TRACE operator commands, see *z/OS MVS System Commands*.

Stopping Processes

There are three ways to stop a process:

- The operator enters a MODIFY operator command to terminate a process.
- A shell user enters the **kill** command to cancel processes.
- The operator enters a CANCEL command to stop an address space containing a process. If the address space contains multiple processes, CANCEL terminates all of the processes.

Terminating a Process with the MODIFY Command

If a process is hung, the operator can enter one of these two MODIFY console commands to terminate the process:

- To allow the signal interface routine to receive control before the process is terminated, issue:

  ```
  F BPXOINIT,TERM=pppp
  ```

  where `pppp` is the process identifier.
- Sometimes a process is not terminated when a TERM request is sent. In these cases, issue:

  ```
  F BPXOINIT,FORCE=pppp
  ```

  where `pppp` is the process identifier.

Terminating a Process with the kill Command

The best way to end a process is to issue the **kill** command. Using the DISPLAY OMVS operator command or the **ps** command, display all the active processes. Then issue the **kill** command, specifying the signal and the PID (process identifier) for the process.

Start by sending a SIGTERM signal:

```
kill -s TERM pid
```

where `pid` is the process identifier. If that does not work, try sending a SIGKILL signal:

```
kill -s KILL pid
```

where `pid` is the process identifier.

Terminating a Process with the CANCEL Command

An operator can cancel all processes or selected processes in an address space. To cancel all processes, use the CANCEL command. Before issuing CANCEL, display all processes running in that address space and the address space identifier by issuing:

```
DISPLAY OMVS,A=xxxx
```

If there is only one process in the address space or if you want to terminate all the processes, issue:

```
CANCEL name,A=asid
```

For example, for a user with a TSO/E user ID of JOE, Figure 19 shows how to obtain the ASIDs for the user's work and then cancel the user's process that is running the **sleep 6000** shell command.

```
display omvs,u=joe
BPXO001I 17.12.23 DISPLAY OMVS 361

OMVS      ACTIVE             OMVS=(93)
USER     JOBNAME  ASID        PID       PPID  STATE   START    CT_SECS
JOE      JOE      001D          5          1  1RI   17.00.10     1.203
JOE      JOE3     001B     131076     262147  1SI   17.00.10      .111
  LATCHWAITPID=          0 CMD=sleep 6000
JOE      JOE1     0041     262147          5  1WI   17.00.10      .595
  LATCHWAITPID=          0 CMD=-sh

cancel joe3,a=1b
```

Figure 19. Console Display for a CANCEL Command

If you want to terminate one or more selected processes in an address space, but not all the processes, then use the MODIFY command as described in "Terminating a Process with the MODIFY Command" on page 33 or the **kill** command as described in "Terminating a Process with the kill Command" on page 33.

Terminating Threads with the MODIFY Command

An operator can terminate a thread, without disrupting the entire process. The syntax of the MODIFY command to terminate a thread is:

```
F BPXOINIT,{TERM}=pid[.tid]
           {FORCE}
```

where

- `pid` indicates the process identifier (PID) of the thread to be terminated. The PID is specified in decimal form as displayed by the D OMVS command.

- `tid` indicates the thread identifier (TID) of the thread to be terminated. The TID is 16 hexadecimal (0-9,A-F) characters as displayed by the following command:

  ```
  D OMVS,PID=pppppppp
  ```

- `TERM=` indicates the signal interface routine will be allowed to receive control before the thread is terminated.
- `FORCE=` indicates the signal interface routine will not be allowed to receive control before the thread is terminated.

  Although abnormal termination of a thread usually causes a process to terminate, using the MODIFY command to terminate a thread will not cause the process to terminate.

You will typically want to terminate a single thread when the thread represents a single user in a server address space. Otherwise, random termination of threads can cause some processes to hang or fail.

If a thread in a process is hung, the operator can enter one of these two MODIFY console commands to terminate the thread without terminating the entire process. We recommend that you use the `TERM` keyword first, and if that does not succeed, use `FORCE`:

- To allow the signal interface routine to receive control before the thread is terminated, use:

  ```
  F BPXOINIT,TERM=pppppppp.tttttttttttttttt
  ```

  where `pppppppp` is the process identifier and `tttttttttttttttt` is the thread identifier.
- To terminate the thread without allowing the signal interface routine to receive control, use:

  ```
  F BPXOINIT,FORCE=pppppppp.tttttttttttttttt
  ```

  where `pppppppp` is the process identifier and `tttttttttttttttt` is the thread identifier.

Shutting Down z/OS UNIX

This section explains how to shut down z/OS UNIX. When you are doing a planned shutdown and will be re-IPLing the system, issue the following operator command:

```
F BPXOINIT,SHUTDOWN=FORKINIT
```

“Planned Shutdowns” on page 36 describes the procedure. If you want to shut down the system as part of JES2 maintenance and do not want to re-IPL the system, use the following operator command:

```
F BPXOINIT,SHUTDOWN=FORKS
```

“Partial Shutdowns (for JES2 Maintenance)” on page 37 describes the procedure.

To shut down file systems as part of a planned shutdown, use the following operator command:

```
F BPXONIT,SHUTDOWN=FILESYS
F BPXONIT,SHUTDOWN=FILEOWNER
```

“File System Shutdown” on page 38 describes the procedure.

Planned Shutdowns

As part of a planned shutdown, you should clean up the system first before re-IPLing.

1. Use the operator SEND command to send a note to all TSO/E users telling them that the system will be shut down at a certain time. For example:

   ```
   send 'The system is being shut down in five minutes. Log off.',NOW
   ```

2. Use the **wall** command to send a similar note about the impending shutdown to all logged-on shell users. For example:

   ```
   wall The system is being shut down in five minutes. Please log off.
   ```

3. Prevent new TSO/E logons and shut down other z/OS subsystems (such as CICS and IMS), following your usual procedures.
4. Shut down all JES initiators.
5. Unmount all NFS-mounted file systems as part of the normal shutdown process.
6. Use normal shutdown procedures to terminate all file system address spaces such as TCP/IP and DFSS. Do this after the final warning has been sent to users that the system is terminating.
7. Terminate running daemons such as **inetd**. To get a list of daemons that are running, issue, for example:

   ```
   D OMVS,U=OMVSKERN
   ```

   In this example, `OMVSKERN` is the user ID that is used for the kernel and daemons. In addition, you can display all processes (most daemons will have recognizable names) by issuing:

   ```
   D OMVS,A=ALL
   ```

   Then use the `F BPXOINIT,TERM=xxxxxxxx` operator command or the **kill** command to terminate those processes.

   7a. Terminate any remaining processes.
8. Move or unmount all file systems (including the root file system) by using the F BPXOINIT,SHUTDOWN=FILEOWNER or F BPXONIT,SHUTDOWN=FILESYS command.
9. Take down JES. At this point, there may still be a number of initiators that are provided by WLM for use on fork and spawn. These initiators time out after 30 minutes on their own. To terminate the initiators, you can issue the following operator command:

   ```
   F BPXOINIT,SHUTDOWN=FORKINIT
   ```

10. After all the processes have been terminated, you can do any of the following:
    - IPL
    - Power off
    - Take down JES, restart JES, and then rebuild your environment. For example:
      - Remount any file systems that you unmounted. To do all the mounts, you must issue mount commands or construct a REXX exec or CLIST. If you are using automount for user file systems, there will be less work involved.
      - If you terminated the address spaces for TCP/IP and DFSS, you must restart these.
      - If you terminated daemons, logon to TSO as superuser and run **/etc/rc** from a shell or from the ISHELL.

– Notify users that the system is once again available for UNIX processing.

Partial Shutdowns (for JES2 Maintenance)

Before JES2 can be shut down for maintenance purposes, part of z/OS UNIX must be shut down. This section explains how you can terminate all of the forked processes without having to re-IPL the entire system. (The kernel remains active but new forked processes are not allowed.) Use this procedure for JES2 maintenance only.

Do the partial shutdown as infrequently as possible because it is a disruptive shutdown; all the user processes that are either forked or non-local spawned are terminated.

After the forked processes have been terminated, you can terminate the colony address spaces. Now JES2 can be shut down for maintenance. z/OS UNIX can be reinitialized after JES2 has been restarted, and forked processes will start being dubbed again. The file system colonies can then be restarted manually. The following steps describe the procedure:

1. Use the operator SEND command to send a note to all TSO/E users telling them that the system will be shut down. For example:

   ```
   send 'The system is being shut down in five minutes. Please log off.'
   ```

2. Use the **wall** command to send a similar note to all logged-on shell users:

   ```
   wall The system is being shut down in five minutes.  Please log off.
   ```

3. Issue the following operator command to begin the shutdown of z/OS UNIX.

   ```
   F BPXOINIT,SHUTDOWN=FORKS
   ```

   This terminates all forked and non-local spawned address spaces on the system. If the operator receives a success message, the shutdown can be continued.

   A failure message means that some forked processes or non-local spawned address spaces could not be terminated. Try to find these processes by issuing:

   ```
   D OMVS,A=ALL
   ```

   To terminate them, issue:

   ```
   F BPXOINIT,FORCE,FORCE=xxxxxxxx
   ```

   If that does not work, use the CANCEL or FORCE operator commands.

4. Terminate the file system colonies that were started under JES (those without SUB=MSTR specified when they were defined).. Use normal shutdown procedures to close these file system address spaces such as Network File System Client (NFSC) and the Distributed File System Cache Manager (DFSCM).

   For NFSC, determine what the process name was used to start this colony. Use this name to cancel it. (For example, `C NFSC`.)

   For DFSCM, use the procedure in *z/OS Distributed File Service DFS Administration* to stop the DFS Cache Manager. Issue `STOP DFSCM` to stop DFSCM.

   For all other colonies, use the procedures documented in their publications.

5. Now you can do whatever corrective or maintenance actions that were needed for JES2, such as restarting it.

6. To restart z/OS UNIX, issue the Modify (F) command.

   ```
   F BPXOINIT,RESTART=FORKS
   ```

7. Restart the file system address spaces.

   For NFSC, you have to respond to the operator message BPXF014D issued when the colony was taken down. Then reissue all the mounts.

   For DFSCM, respond to the operator message BPXF014D.

   For all other colonies, use the procedures they have documented in their product publications.

File System Shutdown

As part of a planned shutdown, you should clean up the file systems before reIPLing by issuing one of the following operator commands:

- F BPXOINIT,SHUTDOWN=FILEOWNER
- F BPXOINIT,SHUTDOWN=FILESYS

This synchronizes data to the file systems and possibly unmounts or moves ownership of the file systems. When SHUTDOWN=FILEOWNER is used, the system will also be disabled as a future file system owner via move or recovery operations until z/OS UNIX has been recycled.

The steps for shutting down z/OS UNIX file systems are the same whether or not the system is participating in shared HFS. However, in a shared HFS environment, the resulting system actions are more complex, because they may involve the movement of file system ownership between systems in the shared HFS group. For more information about the system actions that may occur in a shared HFS environment, see “Shared HFS Implications during a Planned Shutdown of z/OS UNIX” on page 27.

Dynamically Changing the BPXPRMxx Parameter Values

The SETOMVS command enables you to modify BPXPRMxx parmlib settings without re-IPLing. For example:

```
SETOMVS MAXTHREADTASKS=100,MAXPROCUSER=8
```

You can dynamically change process-wide limits separately for each process. For example:

```
SETOMVS PID=123,MAXFILEPROC=200
```

The SET OMVS command enables you to dynamically change the BPXPRMxx parmlib members that are in effect. Because you can have multiple BPXPRMxx definitions, you can easily reconfigure a large set of the system characteristics. You can keep the reconfiguration settings in a permanent location for later reference or reuse. A sample SET OMVS command is:

```
SET OMVS=(AA,BB)
```

If a parameter is specified more than once with different values, in the parmlib members, the first value specified is the first value that is used. For example, if you specify SET OMVS=(AA,BB) where AA has a MAXPROCUSER=10 value and BB has a MAXPROCUSER=5 value, MAXPROCUSER =10 is used.

You can use the SETOMVS RESET command to dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements without having to

re-IPL. However, if you change the values, a re-IPL will be necessary. For more information, see “Dynamically Adding FILESYSTYPE Statements in BPXPRMxx” on page 40.

See *z/OS MVS System Commands* for a complete description of the SET OMVS and SETOMVS commands.

Dynamically Changing Certain BPXPRMxx Parameter Values

The MAXPROCSYS, MAXPTYS, IPCMSGNIDS, MAXFILEPROC, IPCSEMNIDS, IPCSHMNIDS, and IPCSHMSPAGES specify maximum values. You can use the SETOMVS or SET OMVS command to dynamically increase the current system setting, but if you specify a value that is too low or too high, you will get an error message. To use a value outside the range, you must change the specification in BPXPRMxx and re-IPL.

To avoid specifying a value that is too low or too high, you can use a formula to calculate the maximum values. The minimum value is sometimes the current setting of the parameter and sometimes lower than that, as identified in the description of each parameter. The formula for each parameter is described later in this section.

The following example shows you how to perform the calculations using the IPCMSGNIDS parameter, which determines the highest number of unique message queues in the system. To use SETOMVS IPCMSGNIDS=*xxx* to increase the current setting, you must calculate the highest number that you can specify. According to the description of IPCMSGNIDS in “IPCMSGNIDS and IPCSEMNIDS” on page 40, the formula is:

```
MIN(20000,MAX(4096,3*initial value))
```

For this example, the current value of IPCMSGNIDS is 1000; the value of IPCMSGNIDS at IPL is also 1000 (that is, 1000 is the initial value). Use the formula in the following way:

1. Compare 4096 with 3 times 1000 to find the higher number (the MAX). 4096 is the higher number.
2. Compare 20000 with 4096 to find the smaller number (the MIN). 4096 is the smaller number.

Therefore, the highest number that you can specify on SETOMVS IPCMSGNIDS is 4096. The range of numbers that you can specify is 1000 (the current value) to 4096. The correct SETOMVS command for increasing the message queue limit to the maximum (assuming a starting value of 1000) would be:

```
SETOMVS IPCMSGNIDS=4096
```

To change to a number higher than 4096 (but lower than 20000), you will have to change BPXPRMxx and re-IPL.

MAXPROCSYS

The range that you can use has a minimum value of 5; the maximum value is based on the following formula:

```
MIN(32767,MAX(4096,3*initial value)
```

The initial value is the MAXPROCSYS value that was specified during BPXPRMxx initialization. You cannot use a value less than 5. If you want to use a value greater than the current maximum (as calculated by the formula) but lower than the initial maximum (32767), you will have to change the value in BPXPRMxx and re-IPL.

MAXPTYS

The range's minimum value is 1 and the maximum is based on the following formula:

```
MIN(10000,MAX(256,2*initial value)
```

The initial value is the MAXPTYS value that was specified during BPXPRMxx initialization.

IPCMSGNIDS and IPCSEMNIDS

The range's minimum value is the current setting of IPCMSGNIDS or IPCSEMNIDS, and the maximum is based on the following formula:

```
MIN(20000,MAX(4096,3*initial value)
```

The initial value is the value that was specified during BPXPRMxx initialization. If you want to use a value greater than the current maximum (as calculated by the formula) but lower than the initial maximum (20000), you will have to change the value in BPXPRMxx and re-IPL.

IPCSHMNIDS and IPCSHMSPAGES

The range's minimum value is the current setting of IPCMSGNIDS or IPCSHMSPAGES, and the maximum is based on the following formula:

```
MIN(20000,MAX(4096,3*initial value)
```

The initial value is the value that was specified during BPXPRMxx initialization. If you want to use a value greater than the current maximum (as calculated by the formula) but lower than the initial maximum (20000), you will have to change the value in BPXPRMxx and re-IPL.

Dynamically Switching to Different BPXPRMxx Members

Another way to dynamically reconfigure parameters is to use the SET OMVS command to change the BPXPRMxx parmlib members that are in effect. With the SET OMVS command, you can have multiple BPXPRMxx definitions and use them to easily reconfigure a set of the z/OS UNIX system characteristics. You can keep the reconfiguration settings in a permanent location for later reference or reuse.

For example, you could keep the system limits parameters that can be reconfigured in parmlib member BPXPRMLI. When you need to change any of the limits, edit the parmlib member and then issue SET OMVS. For example:

```
SET OMVS=(LI)
```

Changes to system limits (for example, MAXPROCSYS) take effect immediately. Changes to user limits (for example, MAXTHREADS) are set when a new user enters the system (for example, **rlogin** or a batch job). These limits persist for the length of the user connection to z/OS UNIX.

Dynamically Adding FILESYSTYPE Statements in BPXPRMxx

Use the SETOMVS RESET command to dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements without having to re-IPL. If you want to change the values, you will have to edit the BPXPMRxx member that is used for IPLs. You can also dynamically add the parmlib statements currently supported by SETOMVS, such as MAXPROCSYS.

To display information about the current FILESYSTYPE, NETWORK, or SUBFILESYSTYPE statements, issue the following command:

```
DISPLAY OMVS,PFS
```

The following section shows examples of some of the more common configuration changes, adding the HFS and adding sockets. The examples discuss:

1. Activating the HFS file system for the first time.
2. Activating a single sockets file system for the first time.
3. Activating multiple sockets file systems for the first time with Common INET.
4. Adding another sockets file system to an existing common INET configuration.
5. Changing the MAXSOCKETS value.

Activating the HFS File System for the First Time

To activate the HFS file system for the first time, do the following:

1. Set up a root HFS data set.
2. Create a temporary BPXPRM*tt* member that has the following statement:

   ```
   FILESYSTYPE TYPE(HFS) ENTRYPOINT(GFUAINIT)
   ```

3. Issue `SETOMVS RESET=(tt)`.
4. From TSO or the ISHELL, do the following:
   a. Unmount the current root file system.
   b. Mount the root HFS data set as the new root file system.
   c. Mount any additional HFS data sets as needed.
5. Add the following statements to the BPXPRMxx parmlib member used on IPL:
   a. The FILESYSTYPE statement used above.
   b. A ROOT statement for the root HFS.
   c. MOUNT statements for the additional mounts that should be done initially.

Activating a Single Sockets File System for the First Time

This example explains how to activate a single sockets file system for the first time. It uses the SecureWay TCP/IP Socket File System for network sockets and also brings up support for local sockets. The MAXSOCKETS value used is just an example; the value that you use may be different.

1. Create a temporary BPXPRM*tt* member with the following statements:

   ```
   /* Start Address Family AF_INET for Network Sockets /*
   FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
   NETWORK TYPE(INET)    MAXSOCKETS(2000)
      DOMAINNAME(AF_INET)  DOMAINNUMBER(2)

   /* Start Address Family AF_UNIX for Local Sockets  */
   FILESYSTYPE TYPE(UDS)  ENTRYPOINT(BPXTUINT)
   NETWORK TYPE(UDS)      MAXSOCKETS(1000)
     DOMAINNAME(AF_UNIX)  DOMAINNUMBER(1)
   ```

2. Issue `SETOMVS RESET=(tt)`.
3. Start the TCPIP address space.
4. Add these parmlib statements to the BPXPRMxx member used on IPL.

Activating Multiple Sockets File Systems for the First Time with Common INET

This example shows how to activate multiple sockets file systems for the first time with Common INET. It starts two socket file systems, TCP/IP and AnyNet. Because they both support address family AF_INET, they are configured underneath Common INET to give applications the appearance of a single AF_INET socket file system.

Because this is an example of the initial configuration of sockets, the support for local, or AF_UNIX, sockets is also included for completeness.

1. Create a temporary BPXPRM*tt* member with the following statements:

```
/* Start Address Family AF_INET for Common INET */
FILESYSTYPE TYPE(CINET)  ENTRYPOINT(BPXTCINT)
NETWORK TYPE(CINET)    MAXSOCKETS(64000)
     DOMAINNAME(AF_INET)   DOMAINNUMBER(2)
     INADDRANYPORT(5000)  INADDRANYCOUNT(100)
/* Start TCP/IP and AnyNet under Common INET */
SUBFILESYSTYPE TYPE(CINET)  NAME(TCPIP)  ENTRYOINT(EZBPFINI) DEFAULT
SUBFILESYSTYPE TIME(CINET)  NAME(ANYNET) ENTRYPOINT(ISTOEPIT)
```

2. Issue `SETOMVS RESET=(tt)`.
3. Start the TCPIP address space.
4. Start the Sockets Over SNA address space.
5. Add these parmlib statements to the BPXPRMxx member used on IPL.

The names used in the example, `TCPIP` and `ANYNET` must match those used when configuring the associated products.

Increasing the MAXSOCKETS Value

This example shuts down TCP/IP and brings it back up with a new value for MAXSOCKETS:

1. Shut down TCP/IP. For example:

   ```
   p tcpip
   ```

   Most socket programs and daemons will either terminate after TCP/IP is shut down or will tolerate a recycle of TCP/IP. There may be others that will have to be stopped manually.
2. Create a temporary BPXPRM*tt* member that has the following statements:

   ```
   NETWORK TYPE(INET) MAXSOCKETS(10000)
      DOMAINNAME(AF_INET)  DOMAINNUMBER(2)
   ```
3. Issue `SETOMVS RESET=(tt)`.
4. Restart TCP/IP. For example:`S TCPIP`.
5. Restart the socket programs and daemons, as necessary.
6. Update the MAXSOCKETS value in the BPXPRMxx member used on IPL.

Only the SecureWay Socket PFS, EZBPFINI, supports picking up a new MAXSOCKETS value when it is recycled.

The MAXSOCKETS value for a Common INET configuration can be changed with a similar procedure:

1. The TYPE() keyword of the NETWORK statement would specify the TYPE name of the Common INET PFS, which was “CINET” in the previous examples.
2. Common INET is not shut down, though, and the change takes effect in each TCP/IP stack when that stack was recycled.
3. INADDRANYPORT and INADDRANYCOUNT cannot be changed.

Adding Another Sockets File System to an Existing Common INET Configuration

This example starts a second SecureWay Sockets File System and uses names based on the previous examples.

1. Create a temporary BPXPRM*tt* member with the following statements:

   ```
   SUBFILESYSTYPE TYPE(CINET) NAME(TCPIP2) ENTRYPOINT(EZBPFINI)
   ```
2. Issue `SETOMVS RESET=(tt)`.
3. Start the TCPIP2 address space.
4. Add this parmlib statement to the BPXPRMxx member used on IPL.

Tracing Events in z/OS UNIX

To provide problem data, events are traced. When the OMVS address space is started, the trace automatically starts. The trace cannot be completely turned off.

Your installation specifies events to be traced in CTnBPXxx parmlib members. Each member should specify one or more events; keep the number of events small because tracing affects system performance. The installation can filter the events by address spaces, user IDs, and level of detail.

The CTnBPXxx member to be used when the OMVS address space is initialized is identified on the CTRACE parameter of the BPXPRMxx parmlib member. You also specify the size of the trace buffers in the CTnBPXxx member used when the system is IPLed. You can change the buffer size while z/OS UNIX is running. The buffer can be 16KB minimum to 4MB maximum. If you need a different buffer size, change buffer size (BUFSIZE) in a CTnBPXxx member and issue:

```
TRACE CT,ON,COMP=SYSOMVS,PARM=CTnBPXxx
```

An operator starts and stops tracing events in the z/OS UNIX system with the commands:

```
TRACE CT,ON,COMP=SYSOMVS,PARM=CTnBPXxx
TRACE CT,OFF,COMP=SYSOMVS
```

The operator can resume full tracing, with the previously used CTnBPXxx parmlib member or a different member, with the command:

```
TRACE CT,ON,COMP=SYSOMVS,PARM=CTnBPXxx
```

The PARM operand specifies the parmlib member with the tracing options.

Tracing DFSMS/MVS Events

You can also trace DFSMS/MVS events for the HFS. For example, to set up a trace, you can enter the following command:

```
TRACE CT,nnnnk,COMP=SYSSMS
R X,OPTIONS=(CALL,RRTN,CB,SUSP,EXITA,COMP=(ALL,NOIMF,NOSSF)),END
```

or:

```
TRACE CT,nnnnk,COMP=SYSSMS
R X,OPTIONS=(ENTRY,EXIT,EXITA,CB,COMP=(PFS,CDM)),END
```

Attention: SMS trace buffers are allocated in every initiator running kernel workloads. They are allocated in DREF ELSQA, which can cause a shortage of real pages.

For information about how to set up and use a trace, and for diagnosis information on interpreting a trace, see *z/OS DFSMSdfp Diagnosis Reference*.

Re-creating Problems for IBM Service

If you are re-creating a problem for IBM service, it is generally a good idea to increase the OMVS CTRACE buffer size to 4MB. To do this, issue:

```
TRACE CT,4M,COMP=SYSOMVS,PARM=CTnBPXxx
```

with the parmlib member specifying the desired options. Alternatively, you could change the parmlib member to specify the desired buffer size. After you capture the dump for the problem, you can reset the trace buffer size to the original setting. Issue:

```
TRACE CT,xxxK,COMP=SYSOMVS
```

where *xxx*K is the size of the desired trace buffer.

Displaying the Status of the Kernel

Display information about the kernel or processes as follows:

- The operator enters a DISPLAY OMVS command to display the status of the kernel and processes.
- The operator enters the DISPLAY TRACE,COMP=SYSOMVS command to display the status of the kernel trace.
- A shell user enters the **ps** command or the PS ISHELL command to display the status of the user's processes.
- A superuser enters the **ps** command or the PS ISHELL command to display the status of all processes.

The operator displays the status for kernel services with the command:

```
DISPLAY OMVS
```

The command can be used to show information about a user ID, about the parmlib members that are in effect, or about the current values of reconfigurable parmlib member settings.

To display the status of address spaces that the user ID JANES is using and the processor resources used by each address space, the operator enters:

```
DISPLAY OMVS,U=JANES
```

For another example, see Figure 19 on page 34.

If the system IPLed with the specification of OMVS=(XX,YY,ZZ), the output for the D OMVS command is:

```
BPXO004I 10.17.23 DISPLAY OMVS 869
OMVS     ACTIVE  000E         OMVS=(XX,YY,ZZ)
```

The keyword OPTIONS lets you display the current configuration of the BPXPRMxx parmlib statements that are reconfigurable via the SET OMVS or SETOMVS command. The updated output from D OMVS,OPTIONS reflects any changes that resulted from a SETOMVS or a SET OMVS= operator command invocation.

In this example, when the PID option is used to obtain the thread identifiers, the output is:

```
D OMVS,PID=117440514

BPXO040I 14.16.58 DISPLAY OMVS 177
OMVS     000E ACTIVE         OMVS=(93)
USER     JOBNAME  ASID        PID       PPID STATE   START    CT_SECS
MEGA     TC1      0021  117440514  117440515 HKI   14.16.14      .170
  LATCHWAITPID=         0 CMD=ACEECACH
 THREAD_ID        TCB@     PRI_JOB  USERNAME   ACC_TIME SC  STATE
 0496146000000000 009E0438                        .050 PTJ  KU
 04961D0800000001 009D5E88                        .002 SLP JSN
 049625B000000002 009D8798                        .003 SLP JSN
 04962E5800000003 009D5090                        .012 SLP JSN
 0496370000000004 009D5228                        .011 SLP JSN
 04963FA800000005 009D5A88                        .010 SLP JSN
 0496485000000006 009D8048                        .011 SLP JSN
 049650F800000007 009D81E0                        .011 SLP JSN
 049659A000000008 009D8378                        .011 SLP JSN
 0496624800000009 009D8510                        .011 SLP JSN
 04966AF00000000A 009D8930                        .030 SLP JSN
```

You can then cancel selected threads, as shown in this example:

```
F BPXOINIT,FORCE=117440514.04962E5800000003
BPXM027I COMMAND ACCEPTED.

F BPXOINIT,TERM=117440514.0496624800000009
BPXM027I COMMAND ACCEPTED.
```

An operator displays status for the rest of the z/OS system with the commands:

- **DISPLAY TS,LIST**: The number of time-sharing users, including the number of users
- **DISPLAY JOBS,LIST**: The number of active jobs, including the number of address spaces that were forked or that were created in other ways but requested kernel services.
- **DISPLAY A,LIST**: The combined information from the DISPLAY TS,LIST and DISPLAY JOBS,LIST commands.

Displaying the Status of BPXPRMxx Parmlib Limits

You can display information about current system-wide parmlib limits, including current usage and high-water usage, with the DISPLAY OMVS,LIMITS command:

```
DISPLAY OMVS,L
BPXO051I 14.05.52 DISPLAY OMVS 904
 OMVS     0042 ACTIVE          OMVS=(69)
 SYSTEM WIDE LIMITS:         LIMMSG=NONE
                    CURRENT  HIGHWATER     SYSTEM
                      USAGE      USAGE      LIMIT
 MAXPROCSYS               1          4        256
 MAXUIDS                  0          0        200
 MAXPTYS                  0          0        256
 MAXMMAPAREA              0          0        256
 MAXSHAREPAGES            0         10       4096
 IPCMSGNIDS               0          0        500
 IPCSEMNIDS               0          0        500
 IPCSHMNIDS               0          0        500
 IPCSHMSPAGES             0          0     262144 *
 IPCMSGQBYTES           ---          0     262144
 IPCMSGQMNUM            ---          0      10000
 IPCSHMMPAGES           ---          0        256
 SHRLIBRGNSIZE            0          0   67108864
 SHRLIBMAXPAGES           0          0       4096
```

An * displayed after a system limit indicates that the system limit was changed via a SETOMVS or SET OMVS= command.

The display output shows for each limit the current usage, high-water (peak) usage, and the system limit as specified in the BPXPRMxx parmlib member. The displayed system values may be the values as specified in the BPXPRMxx parmlib member, or they may be the modified values resulting from the SETOMVS or SET OMVS commands.

You can also use the DISPLAY OMVS,LIMITS command with the PID= operand to display information about high-water marks and current usage for an individual process.

The high-water marks for the system limits can be reset to 0 with the D OMVS,LIMITS,RESET command. Process limit high-water marks cannot be reset.

Taking a Dump of the Kernel and User Processes

If you have a loop, hang, or wait condition in a process and need a dump for diagnosis, you need to dump several types of data:

- The kernel address space.
- Any kernel data spaces that may be associated with the problem.
- Any process address spaces that may be associated with the problem.
- Appropriate storage areas containing system control blocks (for example, SQA, CSA, RGN, TRT).

The steps are:

1. Use DISPLAY commands to display information on currently active address spaces and data spaces. (For more details on these DISPLAY commands, see *z/OS MVS System Commands*.)
2. Allocate a sufficiently large dump data set.
3. Take the dump.
4. Review the dump completion information.

Displaying the Kernel Address Space

To find the kernel address space and associated data spaces, use `D A,OMVS`. Here is a sample output:

```
D A,OMVS
IEE115I 12.55.47 94.208 ACTIVITY 503
 JOBS     M/S    TS USERS    SYSAS    INITS   ACTIVE/MAX VTAM
00001    00013    00002      00019    00019    00002/00050
 OMVS     OMVS     OMVS     NSW  SO  A=000E   PER=NO   SMC=000
                                     PGN=001  DMN=001  AFF=NONE
                                     CT=033.466S  ET=03.44.48
                                     WUID=STC06055 USERID=OMVSKE
                                     ADDR SPACE ASTE=0173ECC0
                                     DSPNAME=SYSZBPXU ASTE=00A35
                                     DSPNAME=SYSGFU01 ASTE=007F8
                                     DSPNAME=SYSZBPX3 ASTE=007F8
                                     DSPNAME=SYSIGWB1 ASTE=007F8
                                     DSPNAME=SYSZBPX2 ASTE=00A35
                                     DSPNAME=SYSZBPX1 ASTE=00A35
```

The display output shows the kernel address space identifier (ASID) as A=*nnnn* where *nnnn* is the hexadecimal ASID value. In this example, A=000E. The display output also shows the data space names associated with the kernel address space. The system uses these data spaces as follows:

- SYSZBPX1 for kernel data (including CTRACE buffers). The CTRACE buffers are automatically included in the dump and need not be explicitly added to a DUMP command or a SLIP trap.
- SYSZBPX2 for file system data
- SYSZBPX3 for pipes
- SYSIGWB1 for byte-range locking
- SYSGFU01 for file system adapter
- SYSZBPXU for AF_UNIX sockets
- SYSZBPXC for common INET sockets
- SYSZBPXL for local AF_INET sockets

Dump other data spaces if there is reason to believe that they contain data that could be useful in analyzing the problem.

Displaying Process Information

To display the process information for address spaces, use D OMVS,A=ALL. Here is a sample output:

```
   D OMVS,A=ALL

USER     JOBNAME  ASID        PID       PPID STATE
OMVSKERN BPXOINIT 002A          1          0 1WI
MVS      TCPIP    002B      65538          1 MR
DCEKERN  DCEKERN  003A     262147          1 HK
DCEKERN  DCEKERN  003A     262148     262147 HK
DCEKERN  DCEKERN  003A      65541     262147 HK
DCEKERN  DCEKERN  003A      65542     262147 HF
DCEKERN  DCEKERN  003A          7     262147 HK
DCEKERN  DCEKERN  003A          8     262147 HK
TS65106  TS65106  0032          9          1 1RI
TS65106  TS65106  0032         10          9 1CI
  LATCHWAITPID=         0 CMD=-sh
```

The display output shows all of the active processes, ASIDs, process identifiers, parent process IDs, and states. Use this to obtain ASIDs of processes you wish to dump.

Displaying Global Resource Information

To display global resource serialization information to see possible latch contention, use D GRS,C.

This display may show latch contention, which could be the cause of the problem. You should dump the address space of the process holding the latch. If the latch is a file system latch, dump the file system data space SYSZBPX2 also.

Allocating a Sufficiently Large Dump Data Set

Because you are dumping multiple address spaces, multiple data spaces, and multiple storage data areas, you may need a much larger dump data set defined than is normally used for dumping a single address space. You should preallocate a

very large SYS1.DUMPnn data set. For more information on SYS1.DUMPnn data, see the DUMPDS command in *z/OS MVS System Commands*.

SDUMP has a limit on how much storage it allows in a single dump. It is called MAXSPACE. To determine the current value of MAXSPACE, issue the `D D,O` command. The default value is 500 megabytes. To change this value, issue:

```
CD SET,SDUMP,MAXSPACE=nnnnM
```

In a large server environment, you may need to increase MAXSPACE to 2000M (2 gigabytes) or more.

Taking the Dump

To initiate the dump, enter this command:

```
DUMP COMM=(dname)
```

where *dname* is a descriptive name for this dump. You can specify up to 100 characters for the title of the dump. The system responds and gives you a prompt ID. You reply by specifying the data to be included in the dump. If you specify the operand CONT, the system will prompt you for more input.

In the following examples of replies you can give, *rn* is the REPLY number to the prompt.

The data areas in the following reply contain system control blocks and data areas generally necessary for investigating problems:

```
R rn,SDATA=(CSA,SQA,RGN,TRT,GRSQ),CONT
```

In the next reply, `x'E'` is the OMVS address space. The other address space IDs specified are those believed to be part of the problem. You can specify up to 15 ASIDs.

```
R rn,ASID=(E,3A,32),CONT
```

This example specifies data spaces:

```
R rn,DSPNAME=('OMVS'.SYSZBPX2,'OMVS'.SYSZBPX1),END
```

The file system data space, SYSZBPX2, is useful if the hang condition appears to be due to a file system latch.

For more information on the DUMP command, particularly on specifying a large number of operands, see *z/OS MVS System Commands*.

Reviewing Dump Completion Information

After the dump completes, you receive an IEA911E message indicating whether the dump was complete or partial. If it was partial, check the SDRSN value. If insufficient disk space is the reason, delete the dump, allocate a larger dump data set, and request the dump again.

Recovering from a Failure

The operator needs to recover if a failure occurs:

- **Kernel failure:** As a result, interactive processing in the shell and z/OS UNIX applications fail.

- **File system type failure:** z/OS UNIX continues processing even though the file system type is not operational. Requests to use the files in any file systems of that file system type will fail.
- **File system failure:** As a result, some files cannot be used, which may cause programs to fail.

The operator starts recovery by collecting messages and a dump, if written.

System Services Failure

If the z/OS UNIX system fails, the operator collects problem data, which includes messages, SVC dumps, and SYS1.LOGREC records for abends and decides if re-IPL is warranted.

The work in progress when the failure occurred is lost and must be started from the beginning.

File System Type Failure

After a failure of a file system type, the system issues message BPXF014D. In response, the operator or automation corrects the problem as indicated by previous messages and then enters `R` in reply to message BPXF014D.

File System Failure

These events can be symptoms of file system failure:
- 0F4 abend
- EMVSPFSFILE return code
- EMVSPFSPERM return code
- A file becomes unrecognizable or unopenable

After a failure of a file system, the operator:

1. Restores the HFS data set with the data set from the previous level. For more information on recovering an HFS data set, see:
   - *z/OS DFSMS Migration*
   - *z/OS DFSMShsm Storage Administration Guide*
2. Asks a superuser to logically mount the restored HFS data set with a TSO/E MOUNT command.
3. Notifies all shell users that when they invoke the shell they will mount a backlevel file system, telling them the mount point. (Use the **wall** command to broadcast a message to all shell users.)

Files added since the back-level data set was saved must be re-created and added again.

If the physical file system owning the root fails, or if the root file system is unmounted, the operator must restore the root file system. This can be done by a superuser who is defined with a home directory of */*; (root). All work in progress when the failure occurred is lost and must be started from the beginning.

Recovery of DCE Components

Perform any necessary backup of DCE program libraries, configurations, and optional data sets as a part of your regular installation backup and recovery procedures. See *z/OS DCE Administration Guide* for information about DCE recovery.

Managing Interprocess Communication (IPC)

Users can invoke applications that create IPC resources and wait for IPC resources. IPC resources are not automatically released when a process terminates or a user logs off. Therefore, it is possible that an IPC user may need assistance to:

- Remove an IPC resource using the shell's **ipcrm** command
- Remove an IPC resource using the shell's **ipcrm** command to release a user from an IPC wait state

To display IPC resources and which user ID owns the resource, issue the following command:

```
ipcs -w
```

To delete message queue IDs, use the **ipcrm -q** or **ipcrm -Q** command.

Another problem may occur when a user waits a long time for a resource such as semaphores or a message receive. Removing a message queue ID or semaphore ID brings any users in an IPC wait state out of the wait state. To display which users are waiting for semaphores and message queues, issue:

```
ipcs -w
```

Chapter 3. MODIFY Command

Controlling UNIX System Services (z/OS UNIX)

You can use the MODIFY command to control UNIX System Services and to terminate a z/OS UNIX process or thread. You can also use it to shut down z/OS UNIX initiators and to request a SYSMDUMP for a process.

```
F BPXOINIT,{APPL=appl_data}
           {DUMP=pid}
           {FILESYS={DISPLAY[,FILESYSTEM=filesystemname]}[,OVERRIDE]}
                            |,ALL
                            |,EXCEPTION
                            |,GLOBAL
                    {DUMP                                 }
                    {FIX                                  }
                    {REINIT                               }
                    {RESYNC                               }
                    {UNMOUNT,FILESYSTEM=filesystemname    }
                    {UNMOUNTALL                           }
           {FORCE=pid[.tid]}
           {RESTART=FORKS}
           {SHUTDOWN={FILEOWNER | FILESYS | FORKINIT | FORKS}}
           {TERM=pid[.tid]}
```

The parameters are:

BPXOINIT
The name of the job.

APPL=*appl_data*
Allows information to pass straight through to the application. appl_data is a string that is passed back to the invoker in whatever format the application expects it.

DUMP=*pid*
Requests a SYSMDUMP. A SIGDUMP signal is sent to the specified process. *pid* is the decimal form of the process id to be terminated.

FILESYS=
Indicates that a file system diagnostic or recovery operation is to be performed.

This function is applicable only to a sysplex environment where Shared-HFS has been enabled by specifying SYSPLEX(YES) in the BPXPRMxx parmlib member named during system initialization. The command is intended to help diagnose and correct certain Shared-HFS problems or errors that impact one or more systems in a sysplex environment.

Use this command with caution, and only under the direction of an IBM service representative.

To obtain the best results, issue this command at the system with the highest Shared-HFS software service level. To determine which system is executing with the highest Shared-HFS software service level, issue the command

```
F BPXOINIT,FILESYS=DISPLAY,GLOBAL
```

and select the system with the highest “LFS Version” value.

Specify one of the following functions:

DISPLAY *or* **D**
Display the type BPXMCDS couple data set information relating to the Shared-HFS file system. **D** is an alias of **DISPLAY**.

Specify one of the following display options:

ALL
Displays all file systems in the Shared-HFS hierarchy.

EXCEPTION
Displays all file systems that are in an exception state. A file system is in an exception state if one of the following criteria is met:

- State = Mount in progress
- State = Unmount in progress
- State = Quiesce in progress
- State = Quiesced
- State = Unowned
- State = In recovery
- State = Unusable
- The file system state in the couple data set representation is inconsistent with the local file system.

FILESYSTEM=*filesystemname*
Displays information for the specified file system.

GLOBAL
Displays the current sysplex state, consisting of the following items:

- The active systems in the sysplex (system name, logical file system (LFS) version, verification status, recommended recovery action)
- The type BPXMCDS couple data set version number
- The minimum LFS version required to enter the BPXGRP sysplex group
- The name of the system serving BRLM
- The device number of the last mounted file system
- The active ″serialization categories,″ which systems are associated with each category, and the time that each ″serialization category″ was first started. The following serialization categories are defined:
  - SYSTEMS PERFORMING INITIALIZATION
  - SYSTEMS PERFORMING MOVE
  - SYSTEMS PERFORMING QUIESCE
  - SYSTEMS PERFORMING UNMOUNT
  - SYSTEMS PERFORMING MOUNT RESYNC
  - SYSTEMS PERFORMING LOCAL FILE SYSTEM RECOVERY
  - SYSTEMS PERFORMING FILE SYSTEM TAKEOVER RECOVERY
  - SYSTEMS RECOVERING UNOWNED FILE SYSTEMS
  - SYSTEMS PERFORMING REPAIR UNMOUNT

GLOBAL is the default display option.

DUMP
Initiate an SVC dump to capture all of the file system sub-records in the active type BPXMCDS couple data set.

FIX
Perform automatic file system and couple data set diagnosis and repair. As a part of the file system analysis, the system performs an analysis of possible file system latch contention on each system in the sysplex. An operator message identifies any possible problems. The system also analyzes file system serialization data that is maintained in the couple data set, and corrects it if an error is detected. It reports the status of the analysis in an operator message.

Note that the system initiates a dump of critical file system resources as a part of the FIX function. The dump is captured prior to the diagnosis and repair. If, however, a dump was captured due to a FIX or DUMP function that was initiated within the previous 15 minutes, the dump is suppressed.

Perform FIX prior to the UNMOUNTALL and REINIT functions.

REINIT
Re-initialize the file system hierarchy based on the ROOT and MOUNT statements in the BPXPRMxx parmlib member used by each system during its initialization. (Any changes to the BPXPRMxx parmlib member that are made after the system's initialization are not included in the REINIT processing. The system uses the version of the file system parmlib statements that is maintained in kernel storage. It does not re-process the parmlib member.)

Note that the system where the MODIFY command is issued will become the file system server to those file systems common to all systems in the sysplex (such as the ROOT file system) unless the SYSnAME() parameter is specified on the parmlib MOUNT statement.

The intended use of this function is to re-initialize the file system hierarchy after an **UNMOUNTALL** has been performed. However, you can issue **REINIT** at any time; those file systems that are already mounted will not be impacted when **REINIT** processes the parmlib mount statements.

Always issue the FIX function before performing the REINIT function.

RESYNC
Perform a file system hierarchy check on all systems. If a system has not mounted a file system that is active in the Shared-HFS hierarchy, it is mounted locally and thus made available to local applications.

UNMOUNT
Unmount the file system specified by the *filesystem=* parameter. The file system cannot have any active mount points for other file systems. You must unmount those file systems first.

UNMOUNTALL
Unmount all file systems in the sysplex file system hierarchy, including the root file system. When processing is complete, mount SYSROOT on all systems.

Always issue the FIX function before performing the UNMOUNTALL function.

OVERRIDE
Normally only one MODIFY command for a FILESYS= function canbe active on each system. Additionally, only one instance of the MODIFY

command *in the sysplex* can be active for the **FIX, UNMOUNT, UNMOUNTALL,** and **REINIT** functions. If you specify the **OVERRIDE** parameter, the system accepts multiple invocations of this command on each system for the **DISPLAY, DUMP,** and **RESYNC** functions. Note, however, that the second invocation may be delayed.

The primary intent of the **OVERRIDE** parameter is to allow issuance of the DISPLAY functions while there is still a MODIFY in progress and the MODIFY appears to be delayed.

FORCE=
Indicates that the signal interface routine cannot receive control before the thread is terminated.

pid.tid
pid is the decimal form of the process id to be terminated. *tid* is the hexadecimal form of the thread id to be terminated.

RESTART=FORKS
Enables the system to resume normal processing. Suspended dub requests are resumed.

SHUTDOWN=FILEOWNER
Moves or unmounts the z/OS UNIX file systems. The system is disabled as a future file system owner via move or recovery operations until z/OS UNIX has been recycled. New mounts (where this system is designated as the file owner) are not blocked.

SHUTDOWN=FILESYS
Moves or unmounts the z/OS UNIX file systems.

SHUTDOWN=FORKINIT
Shuts down the z/OS UNIX initiators. Normally, these initiators shut themselves down in 30 minutes. Attempts to purge JES2 (command= P JES2) cannot complete until z/OS UNIX initiators have shut down.

SHUTDOWN=FORKS
Requests a shutdown of the fork() service by preventing future forks and non-local spawns. The kernel cannot obtain additional WLM fork initiators for fork and spawn. It attempts to terminate all WLM fork initiator address spaces that are running processes created by fork or non-local spawn. All other services remain “up”, but any new dub requests are suspended until the fork() service is restarted.

TERM=
Indicates that the signal interface routine can receive control before the thread is terminated.

pid.tid
pid is the decimal form of the process id to be terminated. *tid* is the hexadecimal form of the thread id to be terminated.

Example 1

To display process information for a process id of ’117440514’ enter:

```
DISPLAY OMVS,pid=117440514

BPXO040I 14.16.58 DISPLAY OMVS 177
OMVS     000E ACTIVE
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     TC1      0021  117440514  117440515 HKI   14.16.14       .170
  LATCHWAITPID=         0 CMD=ACEECACH
```

```
 THREAD_ID        TCB@     PRI_JOB  USERNAME   ACC_TIME SC  STATE
 0496146000000000 009E0438 OMVS                  .050 PTJ  KU
 04961D0800000001 009D5E88 OMVS     WELLIE1      .002 SLP JSN
 049625B000000002 009D8798 OMVS     WELLIE1      .003 SLP JSN
 04962E5800000003 009D5090 OMVS     WELLIE1      .012 SLP JSN
 0496370000000004 009D5228 OMVS     WELLIE1      .011 SLP JSN
 04963FA800000005 009D5A88 OMVS     WELLIE1      .010 SLP JSN
 0496485000000006 009D8048 OMVS     WELLIE1      .011 SLP JSN
 049650F800000007 009D81E0 OMVS     WELLIE1      .011 SLP JSN
 049659A000000008 009D8378 OMVS     WELLIE1      .011 SLP JSN
 0496624800000009 009D8510 OMVS     WELLIE1      .011 SLP JSN
 04966AF00000000A 009D8930 OMVS     WELLIE1      .030 SLP JSN

 f bpxoinit,force=117440514.04962E5800000003
 BPXM027I COMMAND ACCEPTED.

f bpxoinit,term=117440514.0496624800000009
BPXM027I COMMAND ACCEPTED.
```

Example 2

To shut down the fork() service, enter:

```
F BPXOINIT,SHUTDOWN=FORKS
BPXIxxxE FORK SERVICE HAS BEEN SHUTDOWN SUCCESSFULLY. ISSUE F
BPXOINIT,RESTART=FORKS TO RESTART FORK SERVICE.
```

Example 3

To restart the fork() service, enter:

```
F BPXOINIT,RESTART=FORKS
```

Example 4

Sample outputs of the MODIFY BPXOINIT,FILESYS command:

- F BPXOINIT,FILESYS=DISPLAY,GLOBAL

```
SY1  BPXM027I COMMAND ACCEPTED.
 SY1  BPXF041I 2000/05/12 11.19.18 MODIFY BPXOINIT,FILESYS=DISPLAY,GLOBAL
 SYSTEM   LFS VERSION ---STATUS---------------------- RECOMMENDED ACTION
 SY1        1.  1.  0 VERIFIED                       NONE
 SY2        1.  1.  0 VERIFIED                       NONE
 SY3        1.  1.  0 VERIFIED                       NONE
 CDS VERSION=  1        MIN LFS VERSION=  1.  1.  0
 BRLM SERVER=SY3        DEVICE NUMBER OF LAST MOUNT=      11
 SY1  BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

- F BPXOINIT,FILESYS=DISPLAY,FILESYSTEM=POSIX.SY4.HFS

```
 SY1  BPXM027I COMMAND ACCEPTED.
 SY1  BPXF035I 2000/05/12 11.55.34 MODIFY BPXOINIT,FILESYS=DISPLAY
 -------------NAME-------------------------        DEVICE  MODE
 POSIX.SY4.HFS                                        23  RDWR
   PATH=/SY4
   PARM=SYNC(04)
   STATUS=ACTIVE                LOCAL STATUS=ACTIVE
   OWNER=SY1         RECOVERY OWNER=SY1      AUTOMOVE=Y PFSMOVE=Y
   TYPENAME=HFS      MOUNTPOINT DEVICE=      12
   MOUNTPOINT FILESYSTEM=POSIX.SYSPLEX9.HFS1
   ENTRY FLAGS=90000000  FLAGS=40000000  LFSFLAGS=08000000
   LOCAL FLAGS=40000000  LOCAL LFSFLAGS=2A000000
 SY1  BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

- F BPXOINIT,FILESYS=DISPLAY,FILESYSTEM=POSIX.ZFS.ETC

```
SY1  BPXM027I COMMAND ACCEPTED.
SY1  BPXF035I 2000/05/12 11.55.34 MODIFY BPXOINIT,FILESYS=DISPLAY
-------------NAME-------------------------        DEVICE  MODE
POSIX.ZFS.ETC                                        23  RDWR
  AGGREGATE NAME=POSIX.ZFS.ETC
  PATH=/SY1/etc
  PARM=SYNC(04)
  STATUS=ACTIVE                 LOCAL STATUS=ACTIVE
  OWNER=SY1         RECOVERY OWNER=SY1      AUTOMOVE=Y PFSMOVE=Y
  TYPENAME=ZFS      MOUNTPOINT DEVICE=      12
  MOUNTPOINT FILESYSTEM=POSIX.SYSPLEX9.ZFS1
   AGGREGATE=POSIX.ZFS.ETC
  ENTRY FLAGS=90000000  FLAGS=40000000  LFSFLAGS=08000000
  LOCAL FLAGS=40000000  LOCAL LFSFLAGS=2A000000
SY1  BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

For zFS file systems, the display includes an aggregate file system name, indicating membership in a data set containing multiple file systems. Aggregates provide member file systems with a common pool of disk space.

Appendix A. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS™ enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Volume I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Appendix B. Notices

This information was developed for products and services offered in the U.S.A.

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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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
Mail Station P300
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 information 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

BookManager
CICS
CICS/ESA
DB2
DFS
DFSMS
DFSMSdfp
DFSMSdss
DFSMShsm
DFSMS/MVS
IBM
IBMLink
Language Environment
Library Reader
MVS
NetView
OpenEdition
OS/390
Parallel Sysplex
RACF
Resource Link
SAA
S/390
SecureWay
VTAM
WebSphere
z/OS
z/OS.e
z/Series

Lotus, Domino, and Lotus Go Webserver are trademarks of the Lotus Development Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

NetView is a trademark of International Business Machines Corporation or Tivoli Systems Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

- ANSI (American National Standards Institute)
- IEEE (Institute of Electrical and Electronics Engineers)
- POSIX (Institute of Electrical and Electronics Engineers)
- Tivoli (Tivoli Systems)

Program Number: 5694-A01, 5655-G52

Printed in U.S.A.