z/OS
3.1

UNIX System Services Planning

GA32-0884-60

Note

Before using this information and the product it supports, read the information in “Notices” on page 417.

This edition applies to IBM® z/OS® 3.1 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2024-06-21

Contents

Figures

Tables

About this document

This document presents the information you need to plan for and run an IBM z/OS system with support for z/OS UNIX System Services (z/OS UNIX). This element and the Language Environment® element and z/OS XL C/C++ compiler provide an application programming interface (API) and a shell interface based on the open systems standards of the Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) project, the Federal Information Processing Standard (FIPS), and the X/Open Portability Guide Issue 4 (XPG4).

The z/OS Network File System provides additional capability.

People who run the installation will be able to do the following tasks for z/OS UNIX:

- Customize it.
- Manage operations.
- Manage processing by shell users and application programs.
- Manage file systems.
- Control security.
- Monitor and tune performance.
- Collect data for accounting.

Using this document

This document is for the system programmers, storage administrators, security administrators, and security auditors who run a z/OS system with z/OS UNIX. On other open systems, some system programmer tasks might be done by an administrator.

It assumes the readers are familiar with z/OS systems and its accompanying products.

This document also assumes that you are using Security Server for z/OS. RACF® is a component of the Security Server for z/OS. Instead of RACF, you can use an equivalent security product if it supports the system authorization facility (SAF) interfaces required by z/OS UNIX, which are documented in *z/OS Security Server RACF Callable Services*.

z/OS information

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

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

To find the complete z/OS library, go to IBM Documentation (www.ibm.com/docs/en/zos).

Learning resources: Getting started with IBM zSystems (developer.ibm.com/learningpaths/get-started-ibmz/) provides online learning about IBM Z® basics and Red Hat® Open Shift on IBM Z.

Discussion list

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

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

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

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

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

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

How to provide feedback to IBM

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

Summary of changes

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

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

Summary of changes for z/OS 3.1

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

New

The following content is new.

June 2024 refresh

- The `redirect_dir option` is added to “Mounting a union file system” on page 300. (APAR OA66213, which also applies to z/OS 2.5)

May 2024 refresh

- “IBM z/OS Container Platform” on page 1 is added.

March 2024 refresh

- New environment variables are added. See “Using UNIXPRIV class profiles ” on page 63 and “_BPX environment variables” on page 403. (APAR OA62281, which also applies to z/OS 2.5)
- Additional information is added to “MAXPROCSYS” on page 27. (APAR OA61972, which also applies to z/OS 2.5)
- You can use the prlimit system call to set and retrieve hard and soft resource limits for a specific process. (APAR OA61972, which also applies to z/OS 2.5)

  These sections are updated:

  - “MAXFILEPROC” on page 25
  - “MAXPIPEUSER” on page 26
  - “Superusers in z/OS UNIX” on page 62
  - “What are hard limits?” on page 365
  - “Steps for setting process limits in z/OS UNIX” on page 368
- Chapter 24, “Namespaces for z/OS UNIX,” on page 399 is added. Updates are made to “Using UNIXPRIV class profiles ” on page 63. (APAR OA62731, which also applies to z/OS 2.5)
- Information about the PROC file system is added. See “FILESYSTYPE” on page 21 and Chapter 16, “Managing the PROC file system,” on page 305. (APAR OA62757, which also applies to z/OS 2.5)
- “Nonprivileged container mount and unmount authority” on page 112 is added. (APAR OA62734, which also applies to z/OS 2.5)
- New options are added for the union file system. See “Mounting a union file system” on page 300. (APAR OA62734, which also applies to z/OS 2.5)

September 2023 release

- The FACILITY class profile CONTAINERS is available. For more information, see “Using UNIXPRIV class profiles ” on page 63.

- Information about the union file system (UFS) is added. See Chapter 15, “Managing the union file system (UFS),” on page 299.

Changed

The following content is changed.

June 2024 refresh

- Clarification is added to the description of the _BPXK_AUTOCVT environment variable. See “_BPXK environment variables” on page 405.

May 2024 refresh

- “Increasing the size of the zFS file system” on page 119 is updated.

April 2024 refresh

- Minor updates are made to Chapter 16, “Managing the PROC file system,” on page 305.
- Clarifications are added to “Mounting a union file system” on page 300.
- A new restriction is added to “PID namespaces” on page 399.
- “Shared file system environment” on page 401 is added.

March 2024 refresh

- Information about Unicode Services is added to “Considerations beyond that of Enhanced ASCII” on page 251. (APAR OA65077, which also applies to z/OS 2.4 and 2.5)

November 2023 refresh

- The default for the _BPXK_UNICODE_MAL environment variable is clarified to indicate that NO is the default. See “_BPXK environment variables” on page 405.

September 2023 release

- “Considerations beyond that of Enhanced ASCII” on page 251 is updated with information about multibyte character sets in connection with CCSIDs.

Deleted

The following content was deleted.

September 2023 release

- None

Terminology changes

To ensure alignment with current industry guidelines for inclusive language, the terms *master pseudoterminal* and *slave pseudoterminal* are replaced with *manager pseudoterminal* and *subsidiary pseudoterminal*. As other industry leaders join IBM in embracing the use of inclusive language, IBM will continue to update the documentation to reflect those changes.

Chapter 1. Introduction to z/OS UNIX

The UNIX System Services element of z/OS is a UNIX operating environment, which is implemented within the z/OS operating system. It is also known as z/OS UNIX. The z/OS support enables two open systems interfaces on the z/OS operating system: an application programming interface (API) and an interactive shell interface.

Many users use similar interfaces on other systems and use terminology different from z/OS terminology. For example, they call virtual storage *memory*. The work done by their *system administrators* is handled by system programmers in z/OS systems. Where possible, individual terms and phrases are indicated.

To sum up z/OS UNIX:

- z/OS UNIX System Services (a component of the BCP FMID) provides the following:
  - XPG4 UNIX 1995 conformance
  - Assembler callable services
  - TSO/E commands to manage the file system
  - ISPF shell environment
- z/OS UNIX System Services Application Services (FMID HOTxxxx) interprets commands from users or from programs, called *shell scripts*, and requests MVS™ services in response to the commands. It provides:
  - A TSO/E command to enter the shell environment
  - A shell environment for developing and running applications
  - Utilities to administer and develop in a UNIX environment
  - Support for socket applications
  - Remote login (rlogin) and inetd functions
  - Direct telnet based on TCP/IP protocol
  - A dbx debugger to enable the application programmer to debug source programs that are written in C or C/C++.
  - Support for full-screen applications (curses support)
  - The ability to run programs interactively in the foreground, or in the background

IBM z/OS Container Platform

IBM z/OS Container Platform provides industry-standard cloud technologies that enable you to build your z/OS UNIX applications in container images. IBM z/OS Container Platform includes a container runtime that is built to Open Container Initiative (OCI) standards, which you can use to build and run images as containers natively on z/OS. For more information about IBM z/OS Container Platform, see the following links.

- A content solution, IBM z/OS Container Platform (www.ibm.com/support/z-content-solutions/zos-container-platform/), which contains everything you need to get started. The following IBM article provides an introduction to content solutions: IBM Z and LinuxONE Content Solutions (www.ibm.com/support/z-content-solutions/)
- A Hot Topics article, IBM Brings Containers to z/OS with IBM z/OS Container Platform (ibm.biz/container-platform), which discusses incorporating core, mission-critical z/OS UNIX applications into a container-based cloud-native strategy.

The API interface

With the application programming interface, programs can run in any environment. For example, the programs can run in batch jobs, in jobs submitted by TSO/E users, and in most other started tasks. The programs can also run in any other MVS application task environment. The programs can request only MVS services, z/OS UNIX, or both MVS and z/OS UNIX.

The application interface is composed of C interfaces. Some of the C interfaces are managed within the C/C++ runtime library and others access kernel interfaces to perform authorized system functions on behalf of the unauthorized caller

The interactive shell interface

z/OS UNIX responds to requests from programs and the shells. z/OS UNIX has two shells, the z/OS shell and the tcsh shell. They are collectively called the z/OS UNIX shell. The interactive shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the Restructured eXtended eXecutor (REXX) Language. The shell work consists of:

- Programs run by shell users.
- Shell commands and scripts run by shell users.
- Shell commands and scripts run as batch jobs.

Interacting with elements and features of z/OS

z/OS provides a number of facilities to enable users to interact directly with the operating system. The z/OS UNIX facility enables users to write and invoke shell scripts and utilities, and to use the shell programming language.

z/OS UNIX also interacts with the following elements and features of z/OS:

- BCP (WLM and SMF components).
- Communications Server (TCP/IP).
- Data Facility Storage Management Subsystem (DFSMS).
- Data Set File System (DSFS).
- ISPF, to use the dialogs for OEDIT, OBROWSE, OPUTX, OGETX, or ISPF/PDF for the ISPF shell.
- Language Environment, to execute the shell and utilities or any other XPG4-compliant shell application.
- Network File System (NFS).
- Resource Measurement Facility (RMF).
- Security Server for z/OS. (RACF is a component of the Security Server.)
- System Display and Search Facility (SDSF).
- Time Sharing Option Extensions (TSO/E).
- z/OS File System (zFS).
- z/OS XL C/C++ compiler, to compile programs. In V1R6 and earlier, the compiler was known as the z/OS C/C++ compiler.

Figure 1 on page 3 shows how z/OS UNIX, the shell interface, and the API relate to the rest of the z/OS operating system.

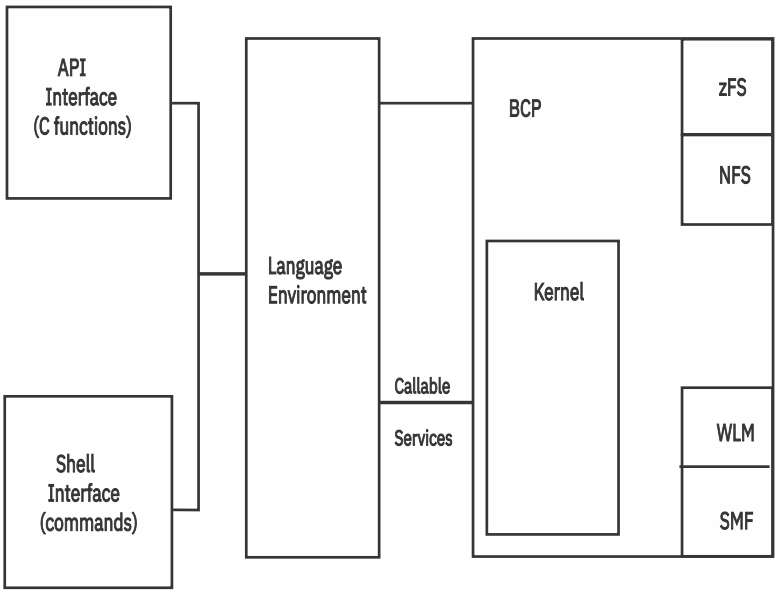


Figure 1. z/OS operating system with z/OS UNIX

Communications Server

Communications Server offers mainframe network security and enables SNA and TCP/IP applications running on z/OS to communicate with partner applications and users on the same or different systems. It enables users to enter the shell environment by using rlogin or telnet from a workstation in the TCP/IP network.

User-written socket applications can use TCP/IP services as a communication vehicle. Both client and server socket applications can use the socket interface to communicate over the Internet (AF_INET and AF_INET6) and between other socket applications by using local sockets (AF_UNIX). An assembler interface is also provided for those applications that do not use the C/C++ runtime library.

For information about multiple transport providers, see Chapter 21, “Setting up for sockets,” on page 377.

Data Set File System (DSFS)

With z/OS Data Set File System (DSFS), z/OS UNIX applications can access data sets by presenting the data sets as a tree-structured file system that is mounted at mount point `/dsfs` in the z/OS UNIX file system tree. A *utility file system* is used by DSFS to contain POSIX information about the data sets accessed by applications to assist in the presentation of data sets as a tree in the z/OS UNIX file system space.

DFSMS

Data Facility System-Managed Storage (DFSMS) manages the data sets used for processing the hierarchical file system which is z/OS. These data sets make up a file hierarchy which consists of files and directories.

- Files contain data or programs. A file containing a load module or shell script or REXX program is called an *executable file*. Files are kept in directories.
- Directories contain files, other directories, or both.

Additional local or remote file systems can be mounted within the file hierarchy.

Interactive System Productivity Facility (ISPF)

Interactive System Productivity Facility (ISPF) offers mainframe network security and enables SNA and TCP/IP applications running on z/OS to communicate with partner applications and users on the same or

different systems. Users of ISPF can use the ISPF shell environment to create, edit, browse, and perform other functions for files and directories in the z/OS UNIX file system.

Language Environment

Language Environment establishes a common language development and execution environment for application programmers on z/OS. To run a shell command or utility, or any user-provided application program written in C or C++, you need the C/C++ runtime library provided with Language Environment.

Network File System (NFS)

Network File System (NFS) enables users to access files on other systems in a network.

Security Server (RACF)

The RACF component of the Security Server authenticates users and verifies whether they are allowed to access certain resources. An equivalent security product can be used to do those tasks.

A user is identified by a UID, which is kept in the OMVS segment of the RACF user profile, and a GID, which is kept in the OMVS segment of the RACF group profile. For more information about OMVS segments, see The OMVS segment in group profiles in *z/OS Security Server RACF Security Administrator's Guide*.

Resource Measurement Facility (RMF)

Resource Measurement Facility (RMF) collects data used to describe z/OS UNIX performance. RMF reports support an address space type of OMVS for address spaces created by fork or spawn callable services and support two swap reason codes. It also monitors the use of resources.

System Management Facilities (SMF)

System management facilities (SMF) is a component of z/OS that provides the ability to run multiple workloads at the same time within one z/OS image or across multiple images. SMF job and job step accounting records identify processes by user, process, group, and session identifiers. Fields in these records also provide information about resources used by the process. SMF file system records describe file system events such as file open, file close, and file system mount, unmount, quiesce, and unquiesce.

Use the JWT, SWT, or TWT value in the SMF parmlib member SMFPRMxx with the BPXPRMxx PWT value to specify when to time out an idle address space that is waiting for terminal activity. For more information, see “SMFPRMxx ” on page 37.

System Display and Search Facility (SDSF)

System Display and Search Facility (SDSF) is an IBM-licensed program that provides a menu-driven full-screen interface that is used to obtained detailed information about jobs and resources in a system. Shell users can enter TSO/E sessions and use SDSF to monitor z/OS activities. For example, they can:

- Monitor printing
- Monitor and control a batch job
- Monitor and control forked address spaces
- Find out which users are logged on to TSO/E sessions

Time Sharing Options Extensions (TSO/E)

Time Sharing Options Extensions (TSO/E) is a licensed program that is based on Time Sharing Option (TSO). With TSO/E, users can interactively share computer time and resources. They can also enter the shell environment by logging on to a TSO/E session and entering the OMVS command. Other TSO/E commands logically mount and unmount file systems, create directories in a file system, and copy files to and from z/OS data sets. Users can switch from the shell to their TSO/E session, enter commands or

do editing, and switch back to the shell. For information about how to perform these tasks using TSO/E commands, see *z/OS UNIX System Services User's Guide*.

WebSphere Application Server Dispatcher

The WebSphere Application Server Dispatcher provides a workload management (WLM) advisor that receives capacity information from WLM and uses it in the load-balancing process. The WLM agent listens on TCP Port 10007 for connection requests from the WLM advisor. The WLM agent is always activated and will attempt to establish a listening socket. To ensure that the WLM agent has access to Port 10007, it must be reserved for the TCP protocol and the job name must be BPXOINIT. In a CINET configuration, the port must be reserved for BPXOINIT on all TCPIP instances.

If the port is not available on one or more TCPIP stacks, the WLM agent is unavailable on all stacks.

To code the port in the TCPIP.PROFILE PORT statement, specify:

```
PORT 10007   TCP   BPXOINIT
```

Workload Management (WLM)

Workload management (WLM) is a component of z/OS that provides the ability to manage multiple workloads at the same time within one z/OS image or across multiple images. When using WLM, you do not need to do any tuning or issue any commands. The kernel uses WLM to create child processes while running in goal mode.

When programs issue fork() or spawn(), the BPXAS PROC found in SYS1.PROCLIB is used to provide a new address space. For a fork(), the system copies one process, called the *parent process*, into a new process, called the *child process*. The forked address space is provided by WLM. Figure 2 on page 5 shows how a fork() creates a new process.

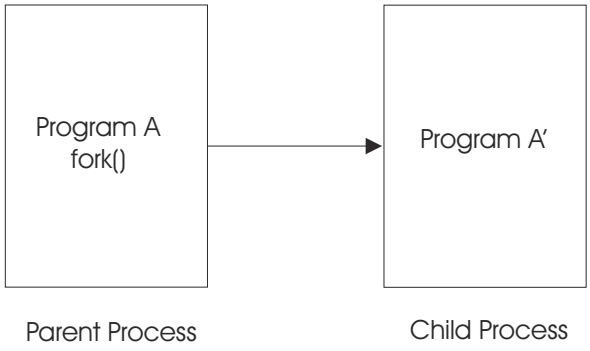


Figure 2. How fork() creates a new process

Existing MVS address space types such as TSO, STC, Batch, and APPC can request z/OS UNIX services. When one of those address spaces makes its first request to the z/OS kernel, the kernel *dubs* the task; that is, it identifies the task as a z/OS UNIX process. There are two types of processes: *user processes*, which are associated with a user, and *daemon processes*, which perform continuous or periodic system-wide functions such as a web server.

Daemons are programs that are typically started when the operating system is initialized and remain active to perform standard services. Some programs are considered daemons that initialize processes for users even though these daemons are not long-running processes. Examples of daemons are as follows:

- **`cron`**, which starts applications at specific times
- **`inetd`**, which provides service management for a network
- **`rlogind`**, which starts a user shell session when one is requested, using a remote rlogin command

In similar systems, initialization typically starts a telnet daemon to perform terminal services.Daemons are not restarted if they stop. You can restart them in any of several ways:

- The z/OS operator can restart daemons using a cataloged procedure. For more information, see "Starting daemons" on page 328.
- A system programmer can restart the daemon from a shell.
- You can use automation products such as Tivoli® NetView® for z/OS to notice daemons terminating and then restart them using cataloged procedures.

A process can have one or more threads; a thread is a single flow of control within a process. Application programmers create multiple threads to structure an application in independent sections that can run in parallel for more efficient use of system resources.

For more information about threads, see pthread_create (BPX1PTC, BPX4PTC) in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

XL C/C++ compiler

C is a programming language designed for a wide variety of programming purposes including system-level code. To compile C code using the c89 command, or to compile C/C++ code using cxx, you need the z/OS XL C/C++ compiler that is available with z/OS.

z/OS File System (zFS)

z/OS File System (zFS) is a UNIX file system. For more information, see "Using the z/OS File System (zFS)" on page 106.

Hardware rules and restrictions for z/OS UNIX

You can use the same hardware as the other components of the z/OS system. Use the same network connections that TSO/E uses and the processor and network connections that JES uses.

- If you want to use rlogin, the connections are different from those for TSO/E users.
- The optional Suppression on Protection feature, if not present, negates certain functions such as mmap().
- For improved TCP/IP performance, install the CHECKSUM hardware.
- To take advantage of improved performance in semaphore processing, you must be running on hardware that supports the PLO (Perform Locked Operation) instruction.

Requirements for accessing kernel services using TSO/E

To access kernel services by using TSO/E, you need the same hardware as other z/OS components. You also need the workstation connections that TSO/E uses and the processor and network connections that JES2 or JES3 uses. Network connections can be made through:

- The Systems Network Architecture network. Configure the workstation hardware and software to access TSO/E through z/OS Communications Server, formerly known as Virtual Telecommunications Access Method (VTAM®).The system requires no additional network definitions for access to z/OS UNIX through TSO/E.
- The TCP/IP network. Configure the workstation hardware and software to communicate with z/OS Communications Server. For the Telnet (TN3270) server, define the Telnet VTAM parameters.
- Rlogin or telnet. For rlogin or telnet, configure the workstation hardware and software to communicate with z/OS Communications Server. If you use rlogin, you might need additional network capacity to support additional rlogin users.

Figure 3 on page 7 shows an example of workstation and network connections for the z/OS system with kernel services.

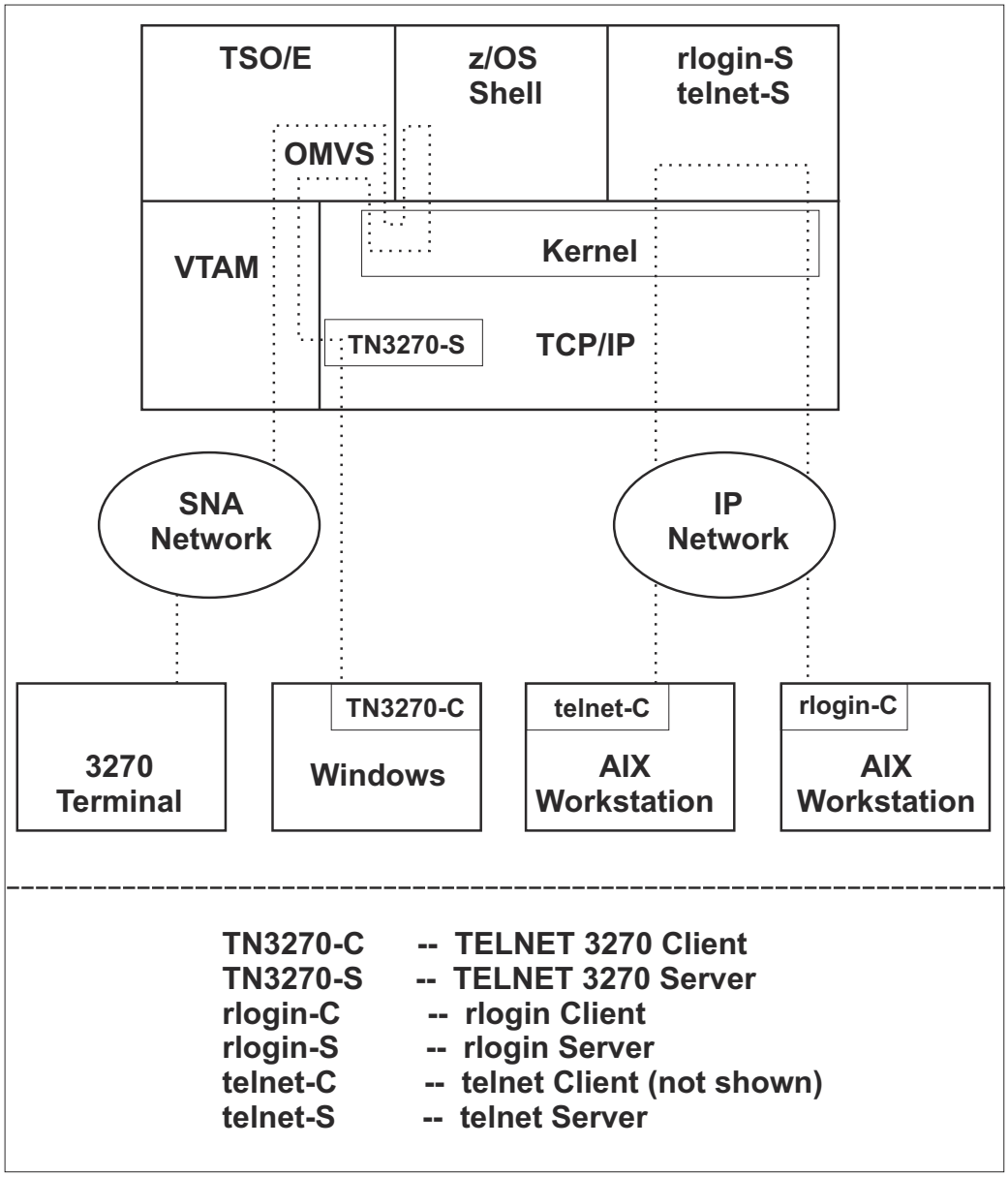


Figure 3. Example of workstation and network connections

Table 1 on page 8 shows several ways that you can access the z/OS UNIX shells:

- The TSO/E OMVS command, which provides a 3270 interface.
- The `rlogin` command, which provides an ASCII interface.

  The `telnet` command, which provides an ASCII interface.

When you first log in to one of the z/OS UNIX shells, you are in line mode. Depending on how you access the shell, you might be able to use utilities that require raw mode (such as `vi`) or run an X Window System application.

Line mode
Input is processed after you press **<Enter>**. Line mode is also called canonical mode.

Raw mode
Each character is processed as it is typed. Raw mode is also called noncanonical mode.

Graphical
A graphical user interface for X Window SystemWindows applications.

Table 1 on page 8 shows different ways of accessing z/OS UNIX from the 3270, workstation, and X Window System terminal.

Table 1. Accessing z/OS UNIX

Terminal	Software at the terminal	Connection to the host	Shell access	Supported modes
3270	Not applicable	Front-end processor such as 3174 or 3172	OMVS (TSO command)	Line
Workstation	3270 emulator (such as pc3270 or tn3270)	Front-end processor such as 3174 or 3172	OMVS (TSO command)	Line
Workstation	rlogin or telnet client	Front-end processor such as 3174 or 3172	rlogin or telnet	Line
Workstation	X-Window server	Front-end processor such as 3174 or 3172	X-Window client	Line or raw
X-terminal	rlogin or telnet client	Front-end processor such as 3174 or 3172	rlogin or telnet	Line or raw
X-terminal	X-Window server	Front-end processor such as 3174 or 3172	X-Window client	Graphical

Noncanonical mode cannot be used with a 3270 because a 3270 does not send data until ENTER, PA, CLEAR, or PF keys are pressed.

Tasks that z/OS UNIX application programmers do

Application programmers are likely to do the following when creating UNIX-compliant application programs:

1. Design, code, and test the programs on their workstations using XPG4 UNIX-conforming systems.
2. Send the source modules from the workstation to z/OS.
3. Copy the source modules from the MVS data sets to z/OS UNIX files.
4. Compile the source modules and link-edit them into executable programs.
5. Test the application programs.
6. Use the application programs.

A z/OS UNIX program can be run interactively from a shell in the foreground or background, run as an MVS batch job, or called from another program.

The following types of applications exist in z/OS UNIX:

- Strictly conforming XPG4-conforming applications
- Applications using only kernel services
- Applications using both kernel and MVS services
- Applications using only MVS services

A z/OS program submitted through the job stream or as a job from a TSO/E session can request kernel services through the following:

- C/C++ functions
- Shell commands, after invoking the shell
- Callable services

At the first request for a kernel service, the system dubs the program as a z/OS UNIX process. C/C++ applications that use RUNOPT 'POSIX(ON)' are always dubbed implicitly. POSIX(OFF) or non-C/C++ applications are not dubbed until an explicit kernel service request is issued.

Administrative tasks using the ISPF shell

The ISPF shell is a panel interface that you can use instead of TSO/E commands or shell commands to perform certain tasks. For example, you can use the ISPF shell to display all mounted file systems or its attributes such as total blocks.

You can also use the ISPF shell to perform the following tasks, which require superuser authority or the RACF SPECIAL attribute or both.

- Create character special files
- Mount a file system
- Unmount a file system
- Reset a pending unmount
- Reset a quiesce status
- Change attributes for z/OS UNIX users
- Display a list of users and sort by name, UID, GID
- Print a list of users
- Set up z/OS UNIX users
- Set up z/OS UNIX groups
- Permit users to alter their own home directory and initial program

For more information about the ISPF shell, see Using the ISPF shell in *z/OS UNIX System Services User's Guide*

Chapter 2. Installing z/OS UNIX

z/OSMF Software Management produces the jobs that are used to install your z/OSMF portable software instance (ServerPac). For CBPDO users, the Program Directory describes how to install your order.

The information that IBM products keep within `/var` is not intended for you to directly edit or modify during the migration period. `/var` is for IBM use and might contain custom-installed configuration files that do not need customer modification during migrations. Like `/etc`, IBM products create directories under `/var` during installation. Unlike `/etc`, IBM products (during execution or customization) also create files in `/var`. However, IBM products do not install files into `/var` during the SMP/E installation.

For migration information, go to *z/OS Upgrade Workflow*.

Methods of installing z/OS UNIX

Two methods of installing z/OS are provided with your z/OS license: z/OSMF portable software instance (ServerPac) and CBPDO. For each of these installation processes, *z/OS Planning for Installation* describes what IBM does for you, what you receive from IBM, and what actions you need to take.

Be familiar with the information that comes with those two installation methods. The information is needed when you install z/OS UNIX, along with the other elements and features. z/OSMF Software Management produces the jobs that are used to install your z/OSMF portable software instance (ServerPac). For CBPDO users, the Program Directory describes how to use the SMP/E RECEIVE, APPLY, and ACCEPT commands to install your order. Both describe the installation verification procedures (IVPs) that you perform to ensure that your installation is proceeding successfully. They also contain customization information.

Installing z/OS UNIX for ServerPac customers

For z/OS ServerPac customers, IBM delivers a single root file system. This file system is unloaded when you do the establishing UNIX Services section of the ServerPac installation process. Not only does the single-root file system make cloning of file systems easier, but it also dramatically reduces the number of jobs that are run by system programmers to establish z/OS UNIX.

Requirement: Performing ServerPac installation requires that you must be a superuser with UID(0) or have access to the BPX.SUPERUSER resource in the FACILITY class. See “Security requirements for ServerPac and CBPDO installation” on page 80 for a complete description of the security requirements necessary to perform your installation.

IBM also delivers a separate file system for `/etc`. See “Establishing an /etc file system for a new release” on page 12.

Installing z/OS UNIX for CBPDO customers

For customers who use the CBPDO (Custom-Built Product Delivery Option) software delivery package, the BPXISZFS sample job in SYS1.SAMPLIB is available. It allocates the root and `/etc` data sets and then mounts them at a given mount point. A file system is allocated and mounted on the `/etc` directory so that z/OS-delivered code is part of a single file system, while customized data can be kept separate. Because there is only one file system, it is easier to clone file systems.

The sample job accepts a mount point directory (commonly referred to as a service directory) to allow you to install new releases of z/OS without affecting your production root file system.

Rule: You must be a superuser with UID(0) or have access to the BPX.SUPERUSER resource in the FACILITY class. See “Security requirements for ServerPac and CBPDO installation” on page 80 for a complete description of the security requirements necessary to perform your install.

Elements and features that install into the file system are installed in both WAVE 1 and WAVE 2 of the CBPDO process and are listed in the *z/OS Program Directory* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Setting up BPXOINIT as a started procedure

BPXOINIT is the started procedure that runs the initialization process. If you are using CBPDO, you have to set up BPXOINIT as a started procedure by adding it to either the RACF STARTED class or the RACF started procedures table, module ICHRINO3, as explained in "Steps for preparing RACF " on page 46.

BPXOINIT is also the job name of the initialization process and is shipped in SYS1.PROCLIB.

The STEPLIB DD statement is propagated from OMVS to BPXOINIT. If there is a STEPLIB DD statement in the BPXOINIT procedure, it is not used if a STEPLIB DD statement was specified in the OMVS procedure.

Establishing an /etc file system for a new release

The `/etc` file system is the location for your own customization data for products. You set up the `/etc` files and you maintain their content. IBM products create *directories* under `/etc` during installation, but IBM does not create *files* under `/etc` during SMP/E installation. Because IBM products do not create files into `/etc`, there is no possibility that SMP/E installation of an IBM product or service will overlay your own files within `/etc`.

Establish the `/etc` file system before you perform the first IPL of the new system. How you establish the directory differs, depending on whether you already have an `/etc` file system.

- If you do not have one, create it using instructions in "Customizing the z/OS UNIX shells " on page 193. For more information about handling files in the `/etc` directory of other z/OS elements and features that install into the z/OS UNIX file system, see *z/OS Upgrade Workflow*. You might also have `/etc` file system impacts for non-z/OS products that you are installing. For that information, see migration and customization information for those products.
- If you already have an `/etc` file system, the `/etc` directory for the new z/OS system is based on a copy of the `/etc` file system for your existing system. You make this copy and migrate your existing `/etc` file system, following instructions in z/OSMF portable software instance (ServerPac) (for those choosing ServerPac) and the Program Directory (for those choosing the CBPDO method of installation). Because the configuration and customization data in your existing `/etc` file system might not be correct for the new system, you might need to make changes to the copy.

Chapter 3. Customizing z/OS UNIX

Before customizing z/OS UNIX, you must decide whether you want to set up kernel services in minimum mode or full function mode. If you want to use any z/OS UNIX service, TCP/IP, or other functions that require the kernel services, you must use full function mode; otherwise, you can use minimum mode.

Requirement: In order to apply service to the file system, you need at least one system that can run in full function mode.

SMS (System Managed Storage, which is part of the DFSMSdfp element of z/OS) must be configured, whether you define the kernel in minimum mode or full function mode.

Setting up kernel services in minimum mode

In minimum mode, the kernel cannot support some functions, such as the z/OS shell and TCP/IP.

If you specify OMVS=DEFAULT in the IEASYSxx parmlib member and then re-IPL, the kernel services start in minimum mode and use the default values for all BPXPRMxx parmlib statements. For information about the default values, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.

In minimum mode, a temporary file system named SYSROOT is used as the root file system. It is initialized and primed with a minimum set of files and directories. Any data written to this file system is not written to DASD. (See Chapter 14, “Managing the temporary file system (TFS),” on page 293 for a description of a temporary file system.) The temporary file system does not have any executables; that is, the shell will not be available. Do not install z/OS UNIX System Services Application Services in the TFS because data will not be written to DASD.

To switch to using kernel services in full function mode, complete the tasks that are described in “Setting up for full function mode” on page 13. The task list in Table 2 on page 13 applies to those who want to use full function mode.

Setting up kernel services in full function mode

If you specify one or more BPXPRMxx members on the OMVS= statement in the IEASYSxx member, then the kernel services start up in full function mode when the system is IPLed. To use the full function mode, you need to perform the tasks listed in Table 2 on page 13.

Before installing the remainder of z/OS, you need to customize SMS, RACF, and the z/OS UNIX file system.

The setup process is listed in “Setting up for full function mode” on page 13.

Setting up for full function mode

The following list identifies the tasks that you do to set up the kernel in full function mode. It is important to follow the order of the tasks.

Some tasks require superuser authority. In that case, that authority might be gained through authority to profiles defined in the UNIXPRIV class that grant certain superuser privileges to users who do not have superuser authority. See “Using UNIXPRIV class profiles ” on page 63 for more information.

Table 2. Task list for customization in full function mode. This table contains a list of tasks for customizing in full function mode.

Task
“Evaluating virtual memory needs” on page 14
Chapter 4, “Establishing UNIX security,” on page 45

Table 2. Task list for customization in full function mode. This table contains a list of tasks for customizing in full function mode. *(continued)*
Task
“Prioritizing UNIX work on your system” on page 15
“Defining the BPXPRMxx members in IEASYSxx” on page 16
“Customizing the BPXPRMxx member of SYS1.PARMLIB” on page 17
“Customizing other members of SYS1.PARMLIB” on page 35
“Initializing the kernel using a cataloged procedure” on page 38
“Running a physical file system in a colony address space” on page 38
“Enabling certain TSO/E commands to z/OS UNIX users” on page 41
Chapter 8, “Customizing the shells and utilities,” on page 191, “Customizing the z/OS UNIX shells ” on page 193
Chapter 17, “Setting up for daemons,” on page 311
“Setting up for rlogin” on page 337
Chapter 18, “Preparing security for servers,” on page 341
“Creating the user file systems” on page 131
Chapter 20, “Tuning performance,” on page 357
“Setting up TCP/IP security” on page 102
Chapter 21, “Setting up for sockets,” on page 377

Checking the mode of the kernel in a running system

To check the mode of the kernel on a running system, issue D OMVS from the console. If the kernel is running in full function mode, you will see output similar to the following:

```
OMVS     000E ACTIVE             OMVS=(2W)
```

where OMVS=(2W) represents the parmlib setting that was used.

If the kernel is running in minimum mode, you will see output similar to the following:

```
OMVS     000E ACTIVE             DEFAULT
```

where DEFAULT indicates that an OMVS setting was not specified in IEASYSxx.

Evaluating virtual memory needs

The kernel services use storage based on expected use as defined by the BPXPRMxx member of SYS1.PARMLIB, as well as by actual use. If you follow the guidelines for Extended Common Service Area (ECSA) and Extended System Queue Area (ESQA), you should avoid running out of storage.

Using extended common service area (ECSA)

The extended common service area (ECSA) is a major storage area above the 16 MB line. It contains pageable system data areas that are addressable by all active virtual address spaces. Use of ECSA is based on the following formulas:

```
#tasks_using_Openmvs * 150 bytes
#processes * 500 bytes
#dubbed_address_space * 500 bytes
```

For example, if your system supports 200 dubbed address spaces, 500 processes, and 2000 threads, the kernel service consumes an additional 650 KB of ECSA.

In addition to this ECSA usage,

- Workload management (WLM) also uses some ECSA for each initiator to satisfy a fork request.
- The OMVS address space uses an additional 20 KB of ECSA. The kernel also uses ECSA to process spawn requests. This storage is freed when no longer needed. Allocate an additional 100 K of ECSA for spawn usage.
- Each process that has a STEPLIB that is propagated from parent to child or across an exec will consume about 200 bytes of ECSA. If STEPLIBs are used for all processes and you have 400 processes, an additional 80 K of ECSA is required.

Extended system queue area (ESQA)

The extended system queue area (ESQA) is a major element of z/OS virtual memory above the 16 MB line. This storage area contains tables and queues that relate to the entire system, and duplicates above the 16 MB line the system queue area (SQA). Kernel services use ESQA in support of several functions. You can use formulas to predict some of the ESQA usage, but others can only be estimated.

The following functions consume ESQA:

1. Signaling uses service request blocks (SRBs) to notify the target of a signal. Signaling frequency is typically not very high and the SRBs are short-lived. For most installations, more ESQA does not need to be allocated in order to support signaling. If you run applications that use signals frequently, increase your ESQA allocation.
2. Using asynchronous socket services causes SRBs to be allocated. Allocate another 100 KB of ESQA if you use asynchronous socket heavily.

Prioritizing UNIX work on your system

Define service classes

You can specify a number of performance periods. Performance periods for short-running work can be given response-time goals or percentile response-time goals. Performance periods for long-running work should be given velocity goals.

1. Define a default service class for forked child processes.
2. Optionally, define additional services classes if you have different work that has different goals (for instance, daemons such as **inetd** versus user work).
3. Define a service class for startup processes, which are forked by the initialization process, BPXOINIT. This service class should be given a velocity goal that is higher than that of other forked child processes.

Define classification rules

Under SUBSYS OMVS:

1. Specify the default service class for forked child processes. All kernel and user forked processes that do not match on any classification rules gets classified in the default service class.
2. Classify startup processes that are forked by the initialization process, BPXOINIT, into their separate service class, classifying by its user ID (USERID=OMVSKERN).
3. Classify other forked child processes that need to be in a separate `srvclass`, classifying based on USERID, ACCTINFO, or TRXNAME.

Under SUBSYS STC: You should allow both OMVS and BPXOINIT to default to the SYSTEM service class. Both the OMVS and BPXOINIT STCs run kernel work and must run with a very high priority. It is recommended that you let these address spaces default to the SYSTEM service class that runs with a dispatching priority of 255. This is especially important in a shared file system environment where cross system communication occurs using XCF services. Running in a service class that does not allow the OMVS address space to be dispatched in a timely manner might prevent the OMVS XCF exits from being dispatched and cause XCF slowdown, which will result in latch contentions.

The following example is a sample service class for forked child processes:

```
- Service Class OMVS - OMVS forked child processes

   Base goal:
    # Duration      Imp       Goal description
    - --------- -   --------  --------------------------------
    1  2000         2          Response Time 80% 1 second
    2  4000         3          Response Time 60% 2 seconds
    3               5          Execution velocity of 10
```

Following is a sample service class for kernel and daemon work:

```
- Service Class OMVSKERN - OMVS startup processes

   Base goal:
    # Duration      Imp       Goal description
    - --------- -   --------  --------------------------------
    1               1          Execution velocity of 40
```

Following is a sample classification for subsystem type OMVS. This sample classifies kernel work into service class OMVSKERN and all other forked work is classified into the default service class named OMVS.

```
- Subsystem Type OMVS

  Classification:

    Default service class is OMVS
    There is no default report class.
      Qualifier  Qualifier          Starting      Service      Report
    # type       name                position      Class        Class
    - ---------- --------------     ---------    --------     -------
    1 UI         OMVSKERN                        OMVSKERN
```

If you have used the PRIORITYGOAL statement in the BPXPRMxx member of SYS1.PARMLIB to enable the nice(), setpriority(), and chpriority() functions, additional service classes for kernel work must be added. For details, see “Controlling dispatching priorities” on page 363.

Defining the BPXPRMxx members in IEASYSxx

After you complete the installation, you need to specify OMVS=*xx* in the IEASYSxx member of SYS1.PARMLIB if you want to start in full function mode. If you do not specify the OMVS parameter or if you specify OMVS=DEFAULT, the kernel is started in minimum mode with all parmlib statements taking their default values. You can specify:

- OMVS=nn, where *nn* is the BPXPRMnn member.

- `OMVS=(nn,mm,...)`, where *(nn,mm,...)* is the set of BPXPRMxx members to use when locating parmlib statements to configure the system services. The first value set for a parameter is the one that is used; if a later member in the list specifies a different value, that value is ignored.

  For example, you have three systems that share parmlib members but do not want to share file systems. Define these parmlib members:

  - BPXPRMLI, which specifies system limits for systems 1 and 2.
  - BPXPRML3, which specifies system limits for system 3, which needs more processes than the other two systems.
  - BPXPRMF1, which specifies file system setup for system 1.
  - BPXPRMF2, which specifies file system setup for system 2.
  - BPXPRMF3, which specifies file system setup for system 3.

  For system 1, the OMVS parameter on the IEASYSxx parmlib member is:

  ```
  OMVS=(F1,LI)
  ```

  For system 2, the OMVS parameter on the IEASYSxx parmlib member is:

  ```
  OMVS=(F2,LI)
  ```

  For system 3, the OMVS parameter on the IEASYSxx parmlib member is:

  ```
  OMVS=(F3,L3)
  ```

  If you want the BPXPRMxx member to be shared by more than one system, you must define system symbols in the IEASYMxx member. Symbols such as system name (&SYSNAME) can be used in BPXPRMxx when referring to file system names.

In order to have different file systems mounted at `/etc` on each system in the sysplex:

```
MOUNT FILESYSTEM('OMVS.&SYSNAME..ETC')
     TYPE(ZFS) MODE(RDWR) MOUNTPOINT (/etc)
```

Customizing the BPXPRMxx member of SYS1.PARMLIB

The BPXPRMxx member of SYS1.PARMLIB contains the parameters that control processing and the file system. This topic only discusses the BPXPRMxx statements that have planning considerations. For a complete list and descriptions of BPXPRMxx statements, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.

You should have two BPXPRMxx members, one defining the values to be used for system setup and the other defining the file systems. Using these two members makes it easier to migrate from one release to another, especially when using the ServerPac method of installation.

When you complete your installation activities, you have one or two BPXPRMxx members, depending on whether you used z/OSMF portable software instance (ServerPac) or CBPDO:

- With z/OSMF portable software instance (ServerPac), you receive two members, as IBM recommends.
- With CBPDO, after you complete all the instructions in the Program Directory, you have the one member that you copied from SYS1.SAMPLIB.

  In this case, you should define a second BPXPRMxx member so that the system setup parameters are in one member and the parameters that define the file systems are in the other.

When you customize the BPXPRMxx members, use columns 1 through 71 for data; columns 72 through 80 are ignored.

Checking the BPXPRMxx syntax

You can use the SETOMVS SYNTAXCHECK operator command to check the syntax of BPXPRMxx before doing an IPL.

The USS_PARMLIB check that is provided by IBM Health Checker for z/OS can be used to determine whether differences exist between current system settings and the settings that are defined in the BPXPRMxx member of SYS1.PARMLIB. If a difference is found, an exception message is issued. You receive a report that lists the differences.

When system and parmlib settings are compared, values with regards to the PARM setting on the MOUNT statement are considered to be case-sensitive. Thus, the same setting value that is expressed in uppercase or lowercase in a system setting are flagged as a difference if that same setting is expressed in the opposite case in the relevant BPXPRMxx member.

For more information about the USS_PARMLIB check, see USS_PARMLIB in *IBM Health Checker for z/OS User's Guide*.

Figure 4 on page 19 shows an example of the IBM-supplied BPXPRMXX member of SYS1.PARMLIB of the current release.

```
MAXPROCSYS(900)
   MAXPROCUSER(25)
   MAXUIDS(200)
   MAXFILEPROC(64000)
   MAXPIPEUSER(8730)
   MAXPTYS(800)
   CTRACE(CTIBPX00)
  /*STEPLIBLIST('/etc/steplib')        */
  /*USERIDALIASTABLE('/etc/tablename') */

/* FILESYSTYPE  TYPE(AUTOMNT)        */
/*              ENTRYPOINT(BPXTAMD)  */

   FILESYSTYPE  TYPE(TFS)
                ENTRYPOINT(BPXTFS)

/* FILESYSTYPE  TYPE(NFS)            */
/*              ENTRYPOINT(GFSCINIT) */
/*              ASNAME(MVSNFSC)      */
/*              PARM('biod(6)')      */

   FILESYSTYPE  TYPE(ZFS)            */
                ENTRYPOINT(IOEFSCM)  */
                ASNAME(ZFS)          */

 FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)
   NETWORK DOMAINNAME(AF_UNIX)
           DOMAINNUMBER(1)

           TYPE(UDS)

   FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
   NETWORK DOMAINNAME(AF_INET)
           DOMAINNUMBER(2)
           MAXSOCKETS(64000)
           TYPE(INET)

/*  NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */
/*   TYPE(INET) */

 /* FILESYSTYPE TYPE(CINET) ENTRYPOINT(BPXTCINT) */
 /* NETWORK DOMAINNAME(AF_INET)                  */
 /*         DOMAINNUMBER(2)                      */
 /*         MAXSOCKETS(64000)                    */
 /*         TYPE(CINET)                          */
 /*         INADDRANYPORT(2000)                  */
 /*         INADDRANYCOUNT(325)                  */
 /* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19)*/
 /*         TYPE(CINET)                          */
 /* SUBFILESYSTYPE NAME(TCPIP)          */
 /*                TYPE(CINET)          */
 /*                ENTRYPOINT(EZBPFINI) */
 /*                DEFAULT              */
```

Figure 4. BPXPRMXX member of SYS1.PARMLIB (Part 1)

```
/* SUBFILESYSTYPE NAME(TCPIP2)         */
/*                TYPE(CINET)          */
/*                ENTRYPOINT(EZBPFINI) */

/* ROOT FILESYSTEM('OMVS.ROOT')
/*              TYPE(ZFS)
/*              MODE(RDWR)
/*              MKDIR('...')

/* MOUNT FILESYSTEM('OMVS.USER.JOE'  )   */
/*              TYPE(ZFS)                */
/*              MODE(RDWR)               */
/*              MOUNTPOINT('/u/joe')     */
/*              NOSETUID                 */
/*              SECURITY                 */
/*              TAG(NOTEXT,0)            */
/*              MKDIR('...'              */


/* ALTROOT FILESYSTEM('OMVS.ALTROOT')      */
/*             MOUNTPOINT('/sysalt')       */
/*             PARM('  ')                  */

  MAXTHREADTASKS(1000)
  MAXTHREADS(200)
/*PRIORITYGOAL (n,...,n) */
/*PRIORITYPG (n,...,n)   */

IPCMSGNIDS    (500)
   IPCMSGQBYTES  (2147483647)
   IPCMSGQMNUM   (10000)
   IPCSHMNIDS    (500)
   IPCSHMSPAGES  (262144)
   IPCSHMMPAGES  (25600)
   IPCSHMNSEGS   (500)
   IPCSEMNIDS    (500)
   IPCSEMNSEMS   (1000)
   IPCSEMNOPS      (25)

   MAXMMAPAREA(40960)
   MAXFILESIZE(NOLIMIT)
   MAXCORESIZE(4194304)
   MAXASSIZE(209715200)
   MAXCPUTIME(1000)
   MAXSHAREPAGES(131072)
   FORKCOPY(COPY)
   SYSPLEX(NO)
   SUPERUSER(BPXROOT)
   TTYGROUP(TTY)
   STARTUP_PROC(OMVS)
 /* STARTUP_EXEC('Dsname(Memname)',SysoutClass) */
 /* RUNOPTS('runtime options') */
   SYSCALL_COUNTS(NO)
   MAXQUEUEDSIGS(1000)
   SHRLIBRGNSIZE(67108864)
   LIMMSG(NONE)
   LOSTMSG(ON)
   PWT(SMF|ENV|SMFENV)
   AUTOCVT(OFF)
   RESOLVER_PROC(DEFAULT)
   SWA(BELOW)
/*SERV_LPALIB('LibraryName',Volser')  */
/*SERV_LINKLIB('LibraryName',Volser') */
  NONEMPTYMOUNTPT(NOWARN) */
  MAXUSERMOUNTSYS(0)      */
  MAXUSERMOUNTUSER(0)     */
/*USERMASK(007)           */
/*SC_EXITTABLE('/etc/scexits') */
```

Figure 5. BPXPRMXX member of SYS1.PARMLIB (Part 2)

You can change some BPXPRMxx values without an IPL.

- The SET OMVS and SETOMVS operator commands dynamically change the settings system-wide. "Dynamically changing the BPXPRMxx parameter values " on page 266 indicates which parameter statements can and cannot be dynamically changed.

- The RACF ALTUSER or ADDUSER commands apply settings on a per-process basis for a particular user, such as Lotus® Domino®. You can use them for the MAXASSIZE, MAXCPUTIME, MAXFILEPROC, MAXPROCUSER, MAXMMAPAREA, and MAXTHREADS parameters.

Defining file systems

You can customize the FILESYSTYPE, ROOT, MOUNT, NETWORK, and SUBFILESYSTYPE statements to define your file systems.

For sharing files across a sysplex, the SYSPLEX(YES) parameter is required, and you must also specify a value for the VERSION statement. See Chapter 7, "Sharing file systems in a sysplex," on page 155 for more information.

FILESYSTYPE

The FILESYSTYPE statement defines the type of physical file system to be used.

When you specify SYSPLEX(YES), you must define the file system type for all systems participating in a shared file system. The easiest way to define it is to have a single BPXPRMxx member that contains file system information for each system participating in a shared file system. If you decide to define a BPXPRMxx for each system, the FILESYSTYPE statement must be identical on each system. See "Customizing BPXPRMxx for a shared file system" on page 165 for more information about configuring BPXPRMxx in a sysplex.

Requirement: Facilities that are required for a particular file system must be initiated on that system. For example, NFS requires TCP/IP, so, if you specify a file system type of NFS, you must also initialize TCP/IP when you initialize NFS, even if there is no network connection.

Table 3 on page 21 lists some types of physical file systems (TYPE parameter) and module names (ENTRYPOINT parameter).

Table 3. Types of file systems. The table lists the file system type and the corresponding module name.

File system type	Description	Module name
AUTOMNT	Handles automatic mounting and unmounting of file systems. The AUTOMNT file system is mounted as AUTOMOVE(YES). However, if the parent file system has the automove unmount attribute, then the automount file system will have that attribute instead of AUTOMOUNT(YES).	BPXTAMD
CINET	Handles requests for the AF_INET and AF_INET6 family of sockets. This enables many different AF_INET or dual AF_INET/AF_INET6 physical file systems to be active on the system. See Chapter 21, "Setting up for sockets," on page 377 for information about setting up sockets. If you want to use CINET, you must be using z/OS Communications Server (TCP/IP Services). If you use CINET, you cannot use INET.	BPXTCINT
INET	Handles requests for the AF_INET and AF_INET6 family of sockets. You must be using z/OS Communication Services (TCP/IP Services). If you use INET, you cannot use CINET.	EZBPFINI
NFS	Handles Network File System requests for access to remote files. For NFS Client, you must create a procedure to run a PFS in a colony address space.	GFSCINIT

Table 3. Types of file systems. The table lists the file system type and the corresponding module name. *(continued)*		
File system type	**Description**	**Module name**
PROC	Manages files that contain process and system information.	BPXUPINT
TFS	Handles requests to the temporary file system (TFS).	BPXTFS
UDS	Handles socket requests for the AF_UNIX address family of sockets.	BPXTUINT
UFS	Obtains merged view of two or more directories.	BPXUNINT
ZFS	Handles z/OS File System requests.	IOEFSCM

Restrictions on VIRTUAL(max)

The VIRTUAL(*max*) value on the FILESYSTYPE PARM('') keyword specifies the maximum amount of virtual memory (in megabytes) that file system data and meta data buffers should use. If you do not specifically set a value for VIRTUAL(*max*), the system assigns to *max* a default value that is equal to half the amount of real storage available to the system at initialization. (The sample BPXPRMxx member provided in SYS1.SAMPLIB uses this default). If you change the storage, consider beforehand how the change will affect your current system storage usage.

You should monitor the paging of your system. If paging is increasing, you might need to set a lower value on the VIRTUAL parameter to relieve the situation.

For more information about VIRTUAL(*max*) and other FILESYSTYPE PARM('') keywords, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.

MOUNT

The MOUNT statement defines the file systems to be mounted at initialization and where in the file hierarchy they are to be mounted. All zFS data sets specified on MOUNT statements in BPXPRMxx must be available at IPL time.

If you specify the NOSETUID option, the setuid and setgid mode bits are not respected when a program in this file system is run. The program runs as though the setuid and setgid mode bits were not set. Also, the APF extended attribute (+a) and the program control extended attribute (+p) are not honored

Rule: The MOUNTPOINT keyword must specify an absolute path name.

Systems using shared file systems will have I/O to a z/OS UNIX couple data set (CDS). Because of these I/O operations to the CDS, each mount request requires additional system overhead. You will need to consider the effect that this change will have on your recovery time if a large number of mounts are required on any system that has shared file systems. For more information about shared file systems, see Chapter 7, “Sharing file systems in a sysplex,” on page 155.

You can use multiple MKDIR keywords on the MOUNT statement to define mount points in BPXPRMxx so that one or more directories is created in the mounted file system during z/OS UNIX initialization.

MKDIR is intended to run during synchronous mounts on the system that is initializing. The directory might not be created if any of these situations exist:

- The file system is mounted asynchronously, such as with NFS.
- The SYSNAME value identifies a remote system.
- The file system is already mounted on a remote system.

Tip: To ensure that the mounts will succeed, use SETOMVS SYNTAXCHECK to check the MVS catalog for the existence of the file system name that is listed on each MOUNT statement.

Additional mount point considerations are as follows:

- Do not include automount-managed directories when you specify mount points in the BPXPRMxx parmlib member. Otherwise mount failures might occur because the automount daemon will start after the parmlib mounts are processed during the IPL.
- Be careful when you add parmlib mount points that include NFS file systems within the path. A hang during IPL might occur if an NFS file system that is not locally served is mounted.

NETWORK

The NETWORK statement defines address families for sockets. It is necessary if the facility needs the socket domains.

If you are activating IPv6, add a second NETWORK statement.

```
FILESYSTYPE Type (INET) Entrypoint (EZBPFINI)
NETWORK DOMAINNAME (AF_INET) TYPE(INET)
        DOMAINNUMBER(2)   MAXSOCKETS(64000)
NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) TYPE(INET)
```

Important: You must configure just AF_INET or both AF_INET and AF_INET6. You cannot configure AF_INET6 alone.

Some tips:

1. You can specify separate MAXSOCKETS values. The default MAXSOCKET value for AF_INET6 is the value that was specified or defaulted to for AF_INET.
2. The INADDRANYPORT range for CINET is shared across both address families and the values are taken from the AF_INET statement. Any value specified on the AF_INET6 statement is ignored.
3. You can also add the second NETWORK statement with SETOMVS RESET, but the TCP/IP stacks will have to be recycled in order to activate IPv6.

ROOT

The ROOT statement defines and mounts the root file system, which must be zFS.

You can use multiple MKDIR keywords on the ROOT statement to define mount points in BPXPRMxx so that one or more directories is created in the mounted file system during z/OS UNIX initialization.

Restriction: MKDIR is intended to run during synchronous mounts on the system that is initializing. The directory might not be created if any of these situations exist:

- The file system is mounted asynchronously, such as with NFS.
- The SYSNAME value identifies a remote system.
- The file system is already mounted on a remote system.

Tip: To ensure that mounts succeed, use SETOMVS SYNTAXCHECK to check the file system name that is listed on each MOUNT statement.

SUBFILESYSTYPE

The SUBFILESYSTYPE statement identifies each of the AF_INET or dual AF_INET/AF_INET6 socket physical file systems that are to run underneath the Common INET socket file system. SUBFILESYSTYPE is an optional statement.

If you plan to support more than one AF_INET or dual AF_INET/AF_INET6 physical file system, such as two TCP/IP networks, the CINET physical file system must be started to manage the multiple file systems

Changes to BPXPRMxx for sockets might also require changes in the user's TCP/IP security system. For more information, see “Setting up TCP/IP security” on page 102.

Defining system limits

You can customize your BPXPRMxx member of SYS1.PARMLIB to provide the performance needed for the way your installation uses kernel services.

Table 4 on page 24 lists the system-wide and process-level limits that can be set in BPXPRMxx. Not all of the statements are explained in this table. For a complete description of each statement, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*. If you specify the SHRLIBMAXPAGES parameter, it will be accepted but will not have any impact on the system. The value that you specify will never be reached, because user-shared library objects are no longer supported.

Table 4. System-wide and process-level limits

System-wide limits	Process-level limits
IPCMSGNIDS	MAXASSIZE
IPCMSGQBYTES	MAXCORESIZE
IPCMSGQMNUM	MAXCPUTIME
IPCSEMNIDS	MAXFILEPROC
IPCSEMNOPS	MAXIOBUFUSER
IPCSEMNSEMS	MAXPROCSYS
IPCSHMMPAGES	MAXPROCUSER
IPCSHMNIDS	MAXQUEUEDSIGS
IPCSHMNSEGS	MAXTHREADS
IPCSHMSPAGES	MAXTHREADTASKS
MAXASSIZE	
MAXMMAPAREA	
MAXPIPES	
MAXPTYS	
MAXUIDS	
MAXUSERMOUNTSYS. The limit is sysplex-wide when it is in the shared file system configuration.	
MAXUSERMOUNTUSER. The limit is sysplex-wide when it is in the shared file system configuration.	
SHRLIBMAXPAGES	
SHRLIBRGNSIZE	

CTRACE

Use the CTRACE statement to provide tracing while the kernel is starting and to avoid having to issue a TRACE operator command to set tracing options.

The only way to change any CTRACE value is with the TRACE command. You cannot use the SETOMVS or SET OMVS command to change the value.

LIMMSG

Use the LIMMSG statement to control the display of console messages that indicate when parmlib limits are reaching critical levels. For more information, see "Displaying the status of system-wide limits specified in BPXPRMxx" on page 275.

MAXASSIZE

MAXASSIZE is the maximum region size (in bytes) for an address space that was created by rlogind, telnetd, and other daemons. The MAXASSIZE value limits the combined size of above and below the 16 M line storage. If MAXASSIZE is greater than LDALIMIT (the <16M limit), then the LDAELIM (the >16M limit) is set to MAXASSIZE - LDALIM.

You can set a system-wide limit in BPXPRMxx and then set higher limits for individual processes. Use the RACF ADDUSER or ALTUSER command to specify the ASSIZEMAX limit on a per-process basis as follows:

```
ALTUSER userid OMVS(ASSIZEMAX(nnnn)
```

MAXCPUTIME

MAXCPUTIME is the time limit (in seconds) for processes that were created by rlogind, telnetd, and other daemons. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the CPUTIMEMAX limit on a per-process basis as follows:

```
ALTUSER userid OMVS(CPUTIMEMAX(nnnn))
```

Specifying a MAXCPUTIME or CPUTIMEMAX of 86400 seconds disables the JWT, SWT, or TWT timeout the same way that JCL TIME=1440 does.

MAXFILEPROC

Use MAXFILEPROC to set the maximum number of file descriptors that a single process can have open concurrently, such as all open files, directories, sockets, and pipes. By limiting the number of open files that a process can have, you limit the amount of system resources a single process can use at one time.

You can use the USS_MAXSOCKETS_MAXFILEPROC check provided by IBM Health Checker for z/OS to determine whether the MAXFILEPROC value is set too low. For more information about the check, see USS_MAXSOCKETS_MAXFILEPROC in *IBM Health Checker for z/OS User's Guide*.

When you select a value, consider these factors:

- For conformance to standards, set MAXFILEPROC to at least 16 to conform to the POSIX standard or at least 25 to conform to the FIPS standard.

  Tip: Set this value to 64000.
- The minimum value of 3 supports stdin, stdout, and stderr.
- The value must be larger than 3 to support shell users. If the value is too small, the shell might issue the message `File descriptor not available`. If this message occurs, increase the MAXFILEPROC value.

A process can change the MAXFILEPROC value by using the setrlimit() or prlimit() function. Only users with appropriate privileges can increase resource limits.

You can set a system-wide limit in BPXPRMxx and then set higher limits for individual processes. Use the RACF ADDUSER or ALTUSER command to specify the FILEPROCMAX limit on a per-process basis as follows.

```
ALTUSER userid OMVS(FILEPROCMAX(nnnn))
```

"Dynamically changing certain BPXPRMxx parameter values" on page 267 explains how to dynamically change the MAXFILEPROC value.

MAXIOBUFUSER

MAXIOBUFUSER limits each user's (for example, a single UID) use of persistent kernel storage for I/O buffers when used in a Unicode Services conversion environment. (See "AUTOCVT" on page 30 or the description of the _BPXK_AUTOCVT environment variable in "_BPXK environment variables" on page 405.) This storage remains allocated for the life of an open file. The amount allocated for each open depends on the CCSID of the file and the size of a read or write requests used by the process. The limit does not apply to UID 0 processes.

z/OS UNIX only tracks such storage for the user that opens the file. If the file is inherited by a different user ID, the amount is not propagated. This can occur, for example, when the user identity changes during spawn or exec. A user identity change is an authorized operation, so the extra tracking is not needed.

MAXMMAPAREA

MAXMMAPAREA specifies the maximum number of data space pages that can be allocated for memory mapping of z/OS UNIX files. Storage is not allocated until memory mappings are active.

For MAXMMAPAREA, you can set a system-wide limit in BPXPRMxx and then set higher limits for individual processes. Use the RACF ADDUSER or ALTUSER command to specify the MMAPAREAMAX limit on a per-process basis. For example:

```
ALTUSER userid OMVS(MMAPAREAMAX(nnnn))
```

The total amount of allocated below the bar mmap pages includes those pages in use by processes limited by the system limit MAXMMAPAREA and also those processes limited by the MMAPAREAMAX process limit for their OMVS segment.

When system limits are being monitored (LIMMSG=SYSTEM/ALL), the BPXI039I resource shortage message is only issued for processes limited by the MAXMMAPAREA value.

Because processes with process limits contribute to the total amount of allocated mmap pages, processes limited by the MAXMMAPAREA value might fail an mmap request before a BPXI039I message is issued. Also, processes with MMAPAREAMAX values for the OMVS segment might be successfully allocating mmap storage even though the BPXI039I message might be displayed with 100% usage.

The SHMEMMAX and MEMLIMIT parameters enable installations to manage the 64-bit space more effectively.

MAXPIPES

The MAXPIPES limit refers to the maximum number of named or unnamed pipes that can be open in the system at any one time. MAXPIPES is a hard system limit and is not configurable. The limit is monitored, and you can view the current usage with the DISPLAY OMVS,LIMITS system command.

For more information about monitoring MAXPIPES, see "Monitoring system and process limits" on page 362.

MAXPIPEUSER

Use MAXPIPEUSER to set the maximum number of named or unnamed pipes that a user (that is, a UID) can open and use concurrently. By limiting the number of pipes that a user can open, you limit the amount of the pipe system resources that a user can use at one time.

The MAXPIPEUSER limit is a hard limit and applies to all users that are not UID=0. The MAXPIPEUSER value cannot be changed on a user basis, and there is no resource limit (setrlimit or prlimit) support to alter a soft or hard limit. For UID=0 users, the MAXPIPEUSER limit of 8,730 (the maximum allowable value) is always enforced.

The MAXPIPEUSER value can be changed dynamically with the SETOMVS command. For more information, see "Dynamically changing the BPXPRMxx parameter values " on page 266.

MAXPROCSYS

MAXPROCSYS specifies the maximum number of processes that can be active at the same time.

You can manage system resources by limiting the number of processes that the system is to support. The values that you specify for MAXPROCSYS, MAXPROCUSER, and MAXUIDS are interrelated. When selecting a value for MAXPROCSYS, remember that these processes are needed:

- The initialization process (BPXOINIT).
- `/usr/sbin/init`, for starting and processing.
- `exec sh` to run a shell script.
- The process in which the shell script runs.

Plan on one process for each daemon (for example, `inetd` and `cron`) that you start from a shell script such as `/etc/rc`. In addition, each shell user needs a minimum of three processes and possibly a few more for piping between shell commands.

Do not specify a higher value for MAXPROCSYS than your system can support because most processes use an entire MVS address space. This value will vary, depending on your environment. If you set the value too high, failures (EAGAIN) for fork or spawn might occur because WLM could not provide enough fork initiators.

Note that the maximum number of namespaces that are allowed on a system is derived from the MAXPROCSYS value. That limit is half of the MAXPROCSYS value at the time that z/OS UNIX was started. The namespace limit is not affected by dynamic changes to the MAXPROCSYS limit, although a shutdown and subsequent restart of z/OS UNIX will update the namespace limit based on the MAXPROCSYS limit at that time.

"Dynamically changing certain BPXPRMxx parameter values" on page 267 explains how to dynamically change the MAXPROCSYS value.

For an example of MAXPROCSYS settings in BPXPRMxx, see "Monitoring use of system resources" on page 362.

MAXPROCUSER

MAXPROCUSER specifies the maximum number of processes that a single user (that is, with the same UID) can have concurrently active.

To improve performance, use MAXPROCUSER to limit user activity.

When selecting a value, consider these factors:

- Set MAXPROCUSER to at least 16 to conform to the POSIX standard for CHILD_MAX, or to at least 25 to conform to the FIPS standard.
- A low MAXPROCUSER value limits the number of concurrent processes that a user can run. A low value limits a user's consumption of processing time, virtual memory, and other system resources.
- Some daemons or users run without UID(0), and might create many address spaces. In these cases, give the daemon ID a high enough PROCUSERMAX value in the OMVS segment.

A user with a UID of 0 is not limited by the MAXPROCUSER value because a superuser might need to be able to log on and use kernel services to solve a problem.

Though not suggested, the security administrator can give the same z/OS UNIX UID to more than one TSO/E user ID. Therefore, the number of users can be greater than the number of UIDs that are defined. Check with the security administrator; if users share UIDs, you will need to define a greater number of processes for each user.

You can set a system-wide limit in BPXPRMxx and then set higher limits for individual processes. Use the RACF ADDUSER or ALTUSER command to specify the PROCUSERMAX limit on a per-process basis. For example:

```
ALTUSER userid OMVS(PROCUSERMAX(nnnn))
```

For an example of MAXPROCUSER settings in BPXPRMxx, see “Monitoring use of system resources” on page 362.

MAXPTYS

Use MAXPTYS to manage the number of interactive shell sessions, where each interactive session requires one pseudo-TTY pair. Do not specify an arbitrarily high value for MAXPTYS.

Note: Because each user might have more than one session, allow four pseudo-TTY pairs for each user (`MAXUIDS * 4`). Specify a MAXPTYS value that is at least twice the MAXUIDS value.

“Dynamically changing certain BPXPRMxx parameter values” on page 267 explains how to dynamically change the MAXPTYS value. For more information about pseudoterminal files, see “Pseudoterminal files” on page 137.

For an example of MAXPTYS settings in BPXPRMxx, see “Monitoring use of system resources” on page 362.

MAXSOCKETS

MAXSOCKETS specifies the maximum number of sockets that can be obtained for a given file system type.

If you are using AF_UNIX, MAXSOCKETS is ignored and the system uses a value of 10000.

You can use the USS_MAXSOCKETS_MAXFILEPROC check provided by IBM Health Checker for z/OS to determine whether the MAXFILEPROC value is set too low. For more information about the health check, see USS_MAXSOCKETS_MAXFILEPROC in *IBM Health Checker for z/OS User's Guide*.

MAXTHREADS

MAXTHREADS is the maximum number of threads that a single process can have active concurrently. If an application needs to create more than the recommended maximum in SAMPLIB, it must minimize storage allocated below the 16 M line by specifying C run-time options. For more information, see set_thread_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread_created threads in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users by using the RACF ADDUSER or ALTUSER command to specify the THREADSMAX limit on a per user basis as follows:

```
ALTUSER userid OMVS(THREADSMAX(nnnn))
```

MAXTHREADTASKS

MAXTHREADTASKS is the maximum number of MVS tasks that a single process can have concurrently active.

A high MAXTHREADTASKS value might affect storage and performance. Each task requires additional storage for the following:

- The control blocks built by the kernel
- The control blocks and data areas required by the Run-Time Library
- System control blocks such as the TCB and RB

You can set a system-wide limit in BPXPRMxx, and then set higher limits for individual users by using the RACF ADDUSER or ALTUSER command to specify the THREADSMAX limits on a per-user basis, as follows:

```
ALTUSER userid OMVS(THREADSMAX(nnnn))
```

MAXUIDS

MAXUIDS specifies the maximum number of unique UIDs that can use kernel services at the same time. The UIDs can be for interactive users or for programs hat requested kernel services.

MAXUIDS limits the number of active UIDs. When you select a value for MAXUIDS, consider these factors:

- Because users are likely to run with three or more concurrent processes each, they require more system resources than typical TSO/E users.
- If the MAXUIDS value is too high relative to the MAXPROCSYS value, too many users can invoke the shell. All users might be affected, because forks might begin to fail.

  For example, if your installation can support 400 concurrent processes— MAXPROCSYS(400)— and each UID needs an average of 4 processes, then the system can support 100 users. For this operating system, specify MAXUIDS(100).

Note: The BPX.DEFAULT.USER profile allowed users without an OMVS segment to be given the default UID. All the users ran with the same UID. In V2R1, BPX.DEFAULT.USER was replaced with BPX.UNIQUE.USER. When BPX.UNIQUE.USER is defined, if a user without an OMVS segment makes a z/OS UNIX syscall, the user is assigned a unique UID. If you were using BPX.DEFAULT.USER and changes to BPX.UNIQUE.USER to assign UIDs to users, this action will result in an increase in the number of UIDs on the system. You will need to increase the MAXUIDS setting accordingly.

For an example of MAXUIDS settings in BPXPRMxx, see "Monitoring use of system resources" on page 362.

MAXUSERMOUNTSYS

Use the MAXUSERMOUNTSYS statement to specify the maximum number of nonprivileged user mounts in the system. For more information about nonprivileged authority, see "Nonprivileged mount and unmount authority" on page 111.

MAXUSERMOUNTUSER

Use the MAXUSERMOUNTUSER statement to specify the maximum number of nonprivileged user mounts allowed for each nonprivileged user. For more information about nonprivileged authority, see "Nonprivileged mount and unmount authority" on page 111.

PRIORITYGOAL

PRIORITYGOAL specifies a list of service class names of 8 characters or less that are used with the nice(), setpriority(), and chpriority() callable services when the system is running in goal mode.

If you are using your system to run a critical real-time application program, specify a list of service class names. It is difficult to run both real-time application programs and general users on the same z/OS UNIX system because you cannot restrict any set of users from access to the nice() and setpriority() functions. For more information, see "Controlling dispatching priorities" on page 363.

PRIORITYPG

PRIORITYPG specifies a list of performance group numbers that are used with the nice(), setpriority(), and chpriority() callable services when the system is running in goal mode.

If you are using your system to run a critical real-time application program, specify a list of performance group numbers. It is difficult to run both real-time application programs and general users on the same z/OS UNIX system because you cannot restrict any set of users from access to the nice() and setpriority() functions. For more information, see "Controlling dispatching priorities" on page 363.

Defining system features

AUTHPGMLIST

AUTHPGMLIST specifies the path name of a z/OS UNIX file that contains the lists of APF-authorized path names and program names. Those lists are called *sanction lists*. For more information about setting up and activating sanction lists, see “Using sanction lists” on page 95. For example:

```
AUTHPGMLIST('/etc/authfile')
```

To dynamically change the value of AUTHPGMLIST, you can use the SETOMVS or SET OMVS =(*xx*) operator command, where *xx* specifies which BPXPRMxx file is to be used to reset the various z/OS UNIX parameters.

If the AUTHGPGMLIST statement contains a nonexistent value, you will not get an error message.

Tip: Using the AUTHPGMLIST statement degrades performance slightly. The more path names or program names that you specify, the greater the performance degradation. However, the tradeoff is increased security.

AUTOCVT

Use the AUTOCVT statement in BPXPRMxx to enable Enhanced ASCII or Unicode Services. When AUTOCVT is set, every read and write operation for a file is checked to see whether any conversion is necessary.

Tip: Use AUTOCVT(OFF). If you want to use another method to enable Enhanced ASCII, see Chapter 11, “Converting files between code pages,” on page 249.

You can also use AUTOCVT(ALL) to enable Unicode Services support. When AUTOCVT(ALL) is set, every read and write operation for a file is checked to see whether conversion is necessary.

LOSTMSG

LOSTMSG(ON) detects lost and duplicate XCF messages in a shared file system configuration. It is ignored if the file system does not have a shared file system configuration; for example, when SYSPLEX(NO) is specified. While LOSTMSG can be specified differently for each member in the sysplex, the same LOSTMSG setting should be specified throughout the sysplex. To disable the detecting of lost and duplicate messages, specify LOSTMSG(OFF).

Tip: Do not use LOSTMSG(ON) when z/OS UNIX sysplex traffic is high, such as when many file systems that are not sysplex-aware are being accessed remotely, because performance might be affected.

LOSTMSG(ON) is the default.

NONEMPTYMOUNTPT

Use the NONEMPTYMOUNTPT statement to control the mounting of file systems on nonempty mount point directories. For more information, see “Restrictions on mounting file systems” on page 113.

PWT

The PWT (process wait time) statement indicates whether processes waiting on terminal input should be timed out. To force a timeout for all processes, set PWT to SMF.

SC_EXITTABLE

Use the SC_EXITTABLE statement to enable exit routines for select z/OS UNIX system calls. By default, when SC_EXITTABLE is not specified, system calls are not enabled for the exits.

Guideline: Using exits degrade performance. The more system calls that are enabled for exits, the greater the performance degradation. When you evaluate performance degradation, consider the following factors:

- How often the system calls are invoked.
- The number of exit routines that were added to the exit points.
- The number of system calls that are enabled for exits.

Requirements: Before the exit routines can be enabled, a syscall exits file must be created that conforms to certain formatting rules. If the file does not follow these formatting rules, the linkage stub name might not be recognized and the system call is not enabled for exits. In addition, IBM recommends that the file reside in the `/etc` directory, for example, `/etc/scexits`. This naming structure fits in with the IBM strategy to place all customized data in the `/etc` directory.

The formatting rules for the syscall exits file are as follows:

- When you include comments in the list, each comment must start with /* and end with */. If the end-of-comment (*/) is not specified, the end of the line is treated as the end of the comment.
- Linkage stub names must be uppercase for them to be recognized correctly.
- Linkage stub names can be on the same line as a comment but not within the comment delimiters.
- Linkage stub names for the same system call can be specified multiple times.
- Either the 31-bit and 64-bit linkage stub name can be specified to identity the callable service to have exits enabled. Exit routines are invoked for the callable service regardless of the AMODE of the service caller.

For example:

```
/*********************************************************************/
/*                                                                   */
/*  Name: Sample system call exits enabling file                     */
/*                                                                   */
/*  Description: Contains a list of z/OS UNIX callable service      */
/*               linkage stub names that identifies the callable     */
/*               services that system call exits are to be enabled   */
/*               for. Callable services with system call exits       */
/*               enabled will invoke all exit routines added to the  */
/*               pre-syscall and/or post-syscall exit points when    */
/*               the callable service is invoked.                    */
/*                                    */
/*               See Appendix K. in the z/OS UNIX System Services    */
/*               Programming: Assembler Callable Service book for    */
/*               details about callable service exits.               */
/*                                     */
/*
*/
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*  Product A - system call enabled for exits                        */
/*                                                                   */
/*********************************************************************/

BPX1STA BPX1FST BPX1RPH BPX1SUI BPX1TLS BPX1GPI BPX1RWD BPX1VOP
BPX4LOD


/*********************************************************************/
/*                                                                   */
/*  Product B - system calls enabled for exits                       */
/*                                                                   */
/*********************************************************************/

BPX1STA /* stat()            */
BPX1FST /* fstat()           */
BPX1RPH /* realpath()        */
BPX1EXC /* exec()            */
BPX1PTC /* pthread_create()  */
BPX1ACC /* access()          */
BPX2ITY /* isatty2()         */
```

Changes to the exits file do not take effect immediately. The table is checked every 15 minutes and refreshed if it was changed. If a change must be activated sooner, you have two options.

- You can use the SETOMVS command.

  ```
  SETOMVS SC_EXITTABLE='/etc/scexits'
  ```

  where `/etc/scexits` is the name of the system call exits file.
- You can also use the SET OMVS command.

  ```
  SET OMVS=xx
  ```

  where *xx* is the name of a z/OS UNIX parmlib member (BPXPRMxx) that contains a SC_EXITTABLE statement. Any other statements that are specified in the BPXPRMxx member are also processed by the SET OMVS command.

After these commands complete, system call exits are enabled for those callable services that are specified in the file. They are disabled for the callable services that are not specified in the file.

SMFUPDATE

The SMFUPDATE statement specifies whether the smf_record() callable service will differentiate between READ and UPDATE access authority to the BPX.SMF.*type*.*subtype* FACILITY class.

SMFUPDATE(ON) allows users with UPDATE authority to the BPX.SMF.*type*.*subtype* FACILITY class to invoke smf_record() without requiring a clean program-controlled environment.

If SMFUPDATE(ON) is not specified in the active BPXPRMxx member, users of smf_record() that are permitted to the BPX.SMF.*type*.*subtype* FACILITY class with any access authority will require a clean program-controlled environment.

SMFUPDATE(OFF) is the default.

STEPLIBLIST

STEPLIBLIST specifies the path name of the file in the file system that contains the list of MVS data sets to be used as step libraries for programs that have the set-user-id and set-group-id bit set on.

Step libraries have many uses; one is so that selected users can test new versions of runtime libraries before the new versions are made available to everyone on the system. Customers who want to do this can put the SCEERUN and SCEERUN2 data set names in this file.

If your installation runs programs that have the setuid or setgid bit turned on, only those load libraries that are found in the STEPLIBLIST sanction list are set up as step libraries in the environment that those programs run in. Because programs with the setuid or setgid bit turned on are considered privileged programs, they must run in a controlled environment. The STEPLIBLIST sanction list provides this control by allowing those programs to use only the step libraries that are considered trusted by the installation.

Tip: The path name of the file should be `/etc/steplib`. This naming strategy fits in with the IBM strategy to place all customized data in the `/etc` directory.

If you do not specify a value for STEPLIBLIST, step libraries will not be set up for set-user-ID and set-group-ID executable files.

These step libraries are set up as a result of the invocation of an executable file by using the exec service (BPX1EXC), the attach_exec service (BPX1ATX) or create (BPX1SPN) service. After one of those services has been invoked, the step libraries can be propagated from the calling task's environment. They can also be specified by using the STEPLIB environment variable that is passed to the exec service. When the exec service invokes a set-user-ID or set-group-ID executable file, only those libraries that are found in the sanctioned list are set up as step libraries in the environment that the executable file will run in.

If the file does not follow these formatting rules, the sanctioned list is not built by using the file.

- You can include comment lines in the list. Each comment line must start with `/*` and end with `*/`.

- You must follow standard MVS data set naming conventions in naming the files in the list.
- Each data set name must be fully qualified and cannot be enclosed in quotation marks.
- Each data set name must be on a line by itself, with no comments.
- You must use uppercase letters for data set names.
- You can put blanks before and after each data set name. Entirely blank lines in the list are ignored.
- You can use the * character to specify multiple files that begin with the same characters. For example, if you list SYS1.*, you are sanctioning any file that begins with SYS1. as a step library.

You should catalog each data set listed in the file to prevent user versions of the data set from being used.

The following sample is a sample-sanctioned list file:

```
/*********************************************************************/
/*                                                                   */
/*   Name: Sample Sanctioned List for set-user-ID and set-group-ID   */
/*         files                                                     */
/*                                                                   */
/*   Updated by:   May only be updated by OSTEPLIB TSO/E command     */
/*                                                                   */
/*   Description:  Contains a list of data set names that may        */
/*                 be used as STEPLIB libraries for SETUID           */
/*                 programs                                          */
/*                                                                   */
/*                 Wild cards may be used to specify multiple        */
/*                 data set names that have the same prefix          */
/*                 characters.                                       */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/* Sanction all data set names beginning with CEE.SCEERUN            */
/*********************************************************************/
CEE.SCEERUN*
```

To create or update the sanctioned list file, use the OSTEPLIB command, which specifies read and run permissions for all users (permissions 555). Because the sanctioned list file must be protected from update by nonprivileged users, only users with superuser authority should be given update access to it.

Updates to the file take effect only when the next setuid(0) program is run from a process with read access to the STEPLIBLIST file because a working copy of the sanctioned list is maintained in storage.

Use the SETOMVS or SET OMVS command to dynamically change the value of STEPLIBLIST. However, this action only changes the current settings of the system. To make a permanent change, edit the BPXPRMxx member that is used for IPLs.

UMASK

The UMASK statement sets a default umask value for all users of z/OS UNIX regardless of how the system address spaces are started or how users sign on to the system.

The UMASK statement applies primarily to a nonshell environment where the umask is not set because /etc/profile was not executed. In a shell environment, /etc/profile is processed when the shell initializes, setting the umask value if it is specified in /etc/profile.

USERIDALIASTABLE

On most UNIX systems, you use lowercase IDs. With z/OS UNIX, typically you will use the uppercase user IDs and group names specified in your security database. In some cases, however, you might want to use lowercase or mixed case names in z/OS UNIX processing. Or perhaps certain names do not conform to your installation's naming conventions. You then need to create an alias table to associate lowercase or mixed case alias names with uppercase z/OS user ID and group names. Note that when lowercase or mixed case alias names are not found in the alias table, they are folded to uppercase.

Using the USERIDALIASTABLE statement degrades performance slightly. The more names that you define, the greater the performance degradation. Installations are encouraged to continue using uppercase-only user IDs and group names defined in their security databases.

Tip: The path name of the file should be `/etc/tablename`. This naming structure fits in with the IBM strategy to place all customized data in the `/etc` directory. If a value for USERIDALIASTABLE is not specified, alias names are not used.

Formatting rules for the alias table:

- You can include comment lines in the list. Each comment line must start with /* and end with */.
- You must follow standard MVS user ID and group name naming conventions in the first column.
- You must follow XPG4 standard naming conventions in the second column.
- Do not enclose the names in quotation marks.
- Each user ID or group name and associated alias name must be on a line by itself, with no comments.
- The MVS user IDs and group names must be located in columns 1-8 and the associated aliases must be located on the same line in columns 10-17.
- The MVS name and the alias name must be separated by 1 or more blanks.
- The tags `:user IDs` and `:groups` must be used to delineate between user IDs and group names.
  - If no tags are present in the file, then all names in the file are assumed to be user IDs.
  - If there are any names listed before a tag, those names are considered to be user IDs.
  - If a `:userids` tag is present, then all name lines following it and up to the next tag are considered to be user IDs.
  - If a `:groups` tag is present, then all name lines following it and up to the next tag are considered to be group names.
  - If specified, the tag must start in column 1.
  - The tag names are not case-sensitive.

If the file does not follow these formatting rules, the alias name might not be recognized and various functions relating to the attempted use of the alias might fail.

The next example is a sample alias table for user IDs and group names.

```
/*****************************************************************/
/*                                                               */
/*   Name: Sample user ID/group name alias table                 */
/*                                                               */
/*   Description:  Contains a list of MVS user IDs and their     */
/*                 associated alias names.                       */
/*                                                               */
/* Alias names can be constructed from uppercase and             */
/* lowercase alphabetic characters. Numbers from 0-9             */
/* can be used as well as the period, underscore, and            */
/* hyphen characters.  Do not use the hyphen as the first        */
/* character.                                                    */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/* Mixed case group names                                        */
/*****************************************************************/
 :Groups
 DEPTD10  DeptD10
 DEPTD20  DeptD20

/*****************************************************************/
/* Non-alphanumeric alias user IDs and group names               */
/*****************************************************************/
 :UserIDs
/*****************************************************************/
/* Mixed case alias names                                        */
/*****************************************************************/
 MYUSERID MyUserid

/*****************************************************************/
```

```
/* Easier to remember alias names                                  */
/*****************************************************************/
 K61XDLBC Daniel

 JOEDOE   Joe_Doe
 MRDOE    Mr.Doe
 ABCD     A-B-C-D
 :groups
 DEVEL    OE-Dev
 TEST     OE_Test
```

For installation security reasons, you might have to use an alias table for user IDs. See "Security requirements for ServerPac and CBPDO installation" on page 80 for more information.

Remember: You must protect the alias table for user IDs and group names. Only users with superuser authority should be given update access to it. All users should be given read access to the file.

Once a user is logged into the system, changing the alias table does not change the alias name immediately. Database queries, however, will yield the new alias if the user ID performing the query has read or execute access to the alias table. The table is checked every 15 minutes and refreshed if it has been changed. If a change needs to be activated sooner, you can use the following command:

```
SETOMVS USERIDALIASTABLE='/etc/tablename'
```

where `/etc/tablename` is the name of the alias table used for the user IDs.

Customizing other members of SYS1.PARMLIB

You might want to customize other members of SYS1.PARMLIB, besides BPXPRMxx.

ALLOCxx

Use the ALLOCxx member of SYS1.PARMLIB to control allocation requests.

Forked address spaces are perceived to be batch jobs by MVS allocation. If a forked address space attempts to allocate a data set on a volume that is not mounted, the request either waits (with or without an operator prompt) or it fails. The ALLOCxx parmlib member controls the behavior of allocation requests of this type. If you do not want the request to wait, specify ALLOCxx statements as follows:

```
VOLUME_ENQ POLICY (CANCEL)
VOLUME_MNT POLICY (CANCEL)
```

Use this policy so that forked addresses do not go into allocation waits. Be aware that using this policy can disrupt your system, because it will cause a failure rather than a wait.

For complete details on using the ALLOCxx parmlib member to prevent waits, refer to in *z/OS MVS Initialization and Tuning Reference*.

COFVLFxx

The virtual lookaside facility (VLF) enables an authorized program to store named objects in virtual storage that is managed by VLF and to retrieve these objects by name on behalf of users in multiple address spaces. VLF is designed primarily to improve performance by retrieving frequently used objects from virtual storage rather than performing repetitive I/O operations from DASD. Certain IBM products or components such RACF use VLF as an alternate way to access data.

If you are using the virtual lookaside facility (VLF), update the VLF parmlib member, COFVLFxx. Add CLASS and EMAJ statements to activate a RACF performance option for z/OS UNIX. The following example shows the added statements in an example of a COFVLF33 member.

```
CLASS NAME(IRRGMAP)          /* GMAP table for z/OS UNIX System Services */
     EMAJ(GMAP)              /* Major name = GMAP                        */
     CLASS NAME(IRRUMAP)     /* UMAP table for z/OS UNIX System Services */
     EMAJ(UMAP)              /* Major name = UMAP                        */
```

Start VLF, specifying the updated member, with the following operator command:

```
START VLF,SUB=MSTR,NN=33
```

For information about caching UIDs and GIDs, see "Caching RACF user and group information in VLF" on page 358.

CTnBPXxx

The CTnBPXxx member of SYS1.PARMLIB is used to control tracing. It specifies the tracing options for a component trace of z/OS UNIX events.

- One member should control initial tracing, which automatically starts when the OMVS address space is started. It should store trace records in a buffer, which could be read if a dump is written. This member should be considered the operating system's default member.

  The CTRACE parameter in the BPXPRMxx member specifies the member; see Figure 4 on page 19, where CTIBPX00 is specified.
- One member can be set up to trace all z/OS UNIX events. This member is CTIBPX01. This method enables a site to change trace information on the fly to obtain suitable component trace information for a dump. (CTIBPX00 traces minimum information.)
- Create other members as needed or when requested by the IBM Support Center.

To change the tracing to collect data needed for a particular problem, ask the operator to enter a TRACE CT command that specifies a different, customized CTnBPXxx member that you have placed in parmlib. When you want to resume normal tracing operations, enter another TRACE CT command specifying the normal CTIBPXxx that your installation uses.

The following example shows the IBM-supplied CTIBPX00 member in SYS1.PARMLIB.

```
TRACEOPTS
          ON
          BUFSIZE(128K)
/*        OPTIONS(                )   */
/*                 'ALL      '    */
/*                ,'CHARS    '    */
/*                ,'DEVPTY   '    */
/*                ,'FILE     '    */
/*                ,'LOCK     '    */
/*                ,'PIPE     '    */
/*                ,'PROCESS  '    */
/*                ,'PTRACE   '    */
/*                ,'SIGNAL   '    */
/*                ,'STK      '    */
/*                ,'STORAGE  '    */
/*                ,'SYSCALL  '    */
/*                ,'DEVRTY   '    */
/*                ,'IPC      '    */
/*                ,'XCF      '    */
/*                 )              */
```

The statements in CTIBPX00 do the following:

- ON turns on the tracing.
- BUFSIZE sets the buffer size at 128KB.

You can specify a buffer size between 16KB and 64MB on the BUFSIZE statement. Use the TRACE CT operator command to change the size of the trace buffer.

Use any of the listed options to specify which events can be traced. For performance reasons, set component tracing off during normal operations. With CTRACE set to OFF, minimal tracing is done.

Customize tracing by adding CTnBPXxx members and storing them in SYS1.PARMLIB. In these members, anticipate events to be traced for diagnosis. The initial CTIBPX00 member specifies minimal tracing. The TRACE CT operator command specifies the customized member name.

The following example shows a member, CTCBPX08, with an OPTIONS statement that requests tracing of events in files and pipes.

```
TRACEOPTS
WTRSTART(CTWTR)
ON
BUFSIZE(4M)
OPTIONS('FILE','PIPE')
WTR(CTWTR)
```

The WTRSTART statement specifies a CTWTR cataloged procedure, which the installation wrote and which starts a component trace external writer. The buffer size is set at 4M.

When re-creating a problem for IBM service, you should increase the buffer size to its maximum.

IEADMR00

The IEADMR00 member of SYS1.PARMLIB is used to gather dump data.

To gather adequate data without an excessive dump size, change IEADMR00 (SYSMDUMP and core dump defaults) to specify:

```
SDATA=(RGN,SUM,TRT,LPA)
```

IKJTSOxx

One of the functions of the IKJTSOxx member of SYS1.PARMLIB is to specify the commands and programs to run on the TSO/E command and program invocation platform. These commands are included as entries in the PLATCMD and PLATPGM statements of that member.

Do not list any of the z/OS UNIX TSO/E commands, including OGET, OPUT, OCOPY, BPXBATCH, MOUNT, UNMOUNT, MKDIR, MKNOD, and OMVS as PLATCMD entries.

SMFPRMxx

To have z/OS shell users be timed out and logged off, you can use the PWT parmlib option, the _BPXK_TIMEOUT environment variable, or the TMOUT environment variable. When PWT is set to SMF, all z/OS UNIX processes waiting in the shell are timed out based on the JWT, TWT, and SWT values of SMFPRMxx.

The PWT and _BPXK_TIMEOUT options will affect shell users waiting in the shell and also processes (such as **vi** or **oedit**) executing in the shell. The TMOUT environment variable will only affect shell users waiting in the shell. When both PWT and TMOUT are specified, TMOUT is only honored when it is less than the JWT, TWT, and SWT values of SMFPRMxx. For example, if SMFPRMxx JWT=(0010) (10 minutes), PWT=SMF (honor the JWT value), and TMOUT=300 (5 minutes), the shell user will be timed out in 5 minutes.

The system administrator can set a TMOUT value in `/etc/profile`. The user can override that value by specifying a TMOUT value in their `.profile`. The TMOUT environment variable contains the number of seconds before user input times out while the z/OS shell is waiting for input at the prompt. If user input is not received, the z/OS shell ends.

To have z/OS shell users be timed out and logged off, you must specify the TMOUT environment variable in `/etc/profile`. The TMOUT environment variable contains the number of seconds before user input times out while the z/OS shell is waiting for input at the prompt. If user input is not received, the z/OS shell ends.

Restriction: The time out value for a TSO user logging into the z/OS UNIX shell with option NOSHAREAS will be more than the JWT value or TWT value when TWT is specified greater than JWT. When z/OS UNIX is started with NOSHAREAS, the **sh** session is in a different address space than the TSO user. If JWT is set to 10 minutes and TWT is set to 20 minutes and there is no terminal activity, the **sh** session will time out in 10 minutes and the TSO user will time out 20 minutes after that.

To have tcsh shell users be timed out and logged off, you must specify the *autologout* variable in `/etc/csh.cshrc` or `/etc/csh.login`. The *autologout* variable contains the number of minutes before user input times out while the tcsh shell is waiting for input at the prompt. If user input is not received, the tcsh shell ends.

Customizing /etc

The `/etc` file system is the location for your own customization data for products. You set up the `/etc` files and you maintain their content. You must copy the files listed in Table 5 on page 38 to the specified directory. These files are used during system initialization.

Table 5. Copying /samples/rc and /samples/init.options to /etc/rc and /etc/init.options

Copied from:	To:
/samples/rc	/etc/rc
/samples/init.options	/etc/init.options

Copying the `/samples/inittab` to `/etc/inittab` is optional.

Table 6. Copying /samples/inittab to /etc/inittab

Copied from:	To:
/samples/inittab	/etc/inittab

Initializing the kernel using a cataloged procedure

A *cataloged procedure* is a set of job control statements that are stored in a system library (for example SYS1.PROCLIB). The storage location for cataloged procedures is installation-defined. For z/OS UNIX, the STARTUP_PROC statement in the BPXPRMxx member of SYS1.PARMLIB specifies a cataloged procedure that initializes the kernel. The default name is OMVS.

Following is the IBM-supplied cataloged procedure in SYS1.PROCLIB:

```
//OMVS PROC
//OMVS EXEC   PGM=BPXINIT,REGION=0K,TIME=NOLIMIT
```

In the EXEC statement in the procedure, the PGM parameter identifies the name of the initialization module. The REGION=0K parameter specifies that the kernel is to use all of the available private area storage within the kernel address space. The TIME=NOLIMIT parameter specifies that kernel is to have unlimited processor time.

Though not recommended, you can replace the OMVS procedure with a procedure that has a different name. If you use a started procedure other than OMVS,

- The replacement started procedure must also be a single job step procedure that invokes the BPXINIT program (EXEC PGM=BPXINIT). If it invokes any other program, OMVS initialization will fail.
- You must change the procedure name in the RACF started procedures table or change the definition in the STARTED Class. See “Steps for preparing RACF ” on page 46.

Started subtasks such as OMVS, BPXOINIT, and colony address spaces fall under SUBSYS STC. These address spaces might be subject to IEFUSI limitations if IEFUSI exits are allowed for SUBSYS STC. IBM strongly recommends that you always set REGION=0 and MEMLIMIT=NOLIMIT for OMVS, BPXOINIT, and colony address spaces.

Running a physical file system in a colony address space

Physical file systems are sometimes initialized in an address space called a *colony address space*. You can think of these address spaces as extensions of the kernel address space. The NFS Client and DFS Client physical file systems must be set up in a colony address space because they must use socket sessions

to talk to their remote servers, which cannot be done from the kernel. You can choose to set up the TFS in a colony address space also; to make that decision see "Running a temporary file system in a colony address space" on page 40.

Some physical file systems cannot be initialized in colony address spaces; for example, the INET or CINET sockets file systems.

Starting colony address spaces

To set up a physical file system in a colony address space, create a cataloged procedure in SYS1.PROCLIB to start the colony address space. Colony address spaces cannot be started using the START operator command.

The name of the procedure must match the name specified on an ASNAME operand on the FILESYSTYPE statement in BPXPRMxx that starts physical file systems in this colony address space.

For example, an NFS client with the cataloged procedure NFSCLNT is associated with the following FILESYSTYPE statement:

```
FILESYSTYPE TYPE (NFS)
    ENTRYPOINT(GFSCINIT)
    ASNAME(NFSCLNT)
```

The procedure must contain the statement:

```
EXEC PGM=BPXVCLNY
```

Starting colony address spaces outside of JES

If you do not want colony address spaces to be started under JES (which is the default), you can change this by including the SUB=MSTR parameter with the ASNAME keyword. The ASNAME keyword is specified as:

```
ASNAME(procname,'start_parms')
```

where:

- The first value is required and is a 1-to-8-character name in SYS1.PROCLIB.
- The second value is optional and is a quoted string that is appended to the procname when the address space is started. The string can be up to 100 characters long.

The start_parms are not validated; they are just passed to the system when the address space is started with an internal start command as in `procname,start_parms`. For example:

```
ASNAME (NFSCLNT,'SUB=MSTR')
```

The colony address space runs outside of JES control and does not have to be stopped if JES has to be stopped, which facilitates planned shutdowns of individual systems in a sysplex that has shared file systems. The NFS Client, TFS, and zFS physical file systems support running outside of JES and the following information might help you to decide whether to move these z/OS UNIX colonies outside of JES. The DFS Client PFS does not support being started outside of JES.

z/OS UNIX colony address spaces are started procedures. If you do not want to run them under JES, you will need to change any DD SYSOUT= data sets that are specified in these procedures. These must be changed because SYSOUT data sets are only supported under JES. There are three ways you can change these data sets:

1. Direct the output to a named data set by changing to DD DSN=.
2. Direct the output to a named file by changing to DD PATH=.
3. Throw the output away by changing to DD DUMMY.

Restriction: If the NFS or zFS colony address space is started at IPL time, then PATH= cannot be used because the MOUNT statements have not been processed yet.

Additionally, there are some DD names that Language Environment will open under certain conditions. If these data sets have not been allocated in the procedure, Language Environment dynamically allocates them with SYSOUT=. The DD names are:

SYSIN
For standard input.

SYSPRINT
For standard output. If SYSPRINT does not exist, Language Environment looks for SYSTERM or SYSERR. If one of those exists, it will be used. But Language Environment does not dynamically allocate either SYSTERM or SYSERR.

SYSOUT
For standard error. It is also the default message file DD.

CEEDUMP
For capturing dumps formatted by Language Environment.

If any of these names are not currently used in the colony procedure, you must add them with DD DUMMY.

If any of the existing DD SYSOUT= statements are not changed, or any of those dynamically allocated by Language Environment are not added, and an attempt is made to open that DD name, the result will be an ABENDS013. Exactly which DD names are opened and when varies by name and product and the situation.

There are also other consequences of running outside of JES that you might need to consider:

- SDSF displays will not list the colony address space.
- There will be no job log or system messages data set.
- System messages will go to SYSLOG.
- SMF recording is different between JES and the master subsystem.

For information about setting up security for the colony address space, see Step “6” on page 49.

Running a temporary file system in a colony address space

In some situations, you might want to run a temporary file system in a colony address space instead of the kernel address space. Because the temporary file system can use up a large amount of kernel virtual memory, there might be some environments in which the kernel can run out of private storage. This can happen on large systems with many shell users or in some Lotus environments. By putting the temporary file system in a colony, impact on the kernel is reduced, and you can have a larger temporary file system.

Restriction: Because TFS uses virtual memory, your real and auxiliary storage configurations must be large enough to accommodate the size of all of the temporary file systems that you mount.

Steps for creating a cataloged procedure for a temporary file system

Perform the following steps to create a cataloged procedure for a temporary file system.

1. Set up a FILESYSTYPE statement in BPXPRMxx. It must have the name of the procedure on the ASNAME operand.

   For cataloged procedure XXXXXX, the FILESYSTYPE statement would be:

   ```
   FILESYSTYPE TYPE(TFS)
               ENTRYPOINT(BPXTFS)
               ASNAME(XXXXXX)
   ```

   _____

2. Create the cataloged procedure XXXXXX in SYS1.PROCLIB. It must contain the following statement:

   ```
   EXEC PGM=BPXVCLNY
   ```

_____

When you are done, you have created a cataloged procedure for a temporary file system.

Some tips:

1. A temporary file system uses private storage for the file system in memory. If you run it in the kernel, then you might run out of virtual memory. However, by starting multiple temporary file systems in colonies, you can create many temporary files or very large temporary files (about 1.5 gigabytes per temporary file system colony).
2. Code REGION=0K, REGIONS=0M, or a specific MEMLIMIT value on the EXEC statement in the TFS colony cataloged procedure. Doing so allows the TFS address space to address storage above the bar for the TFS file system. For more information about the MEMLIMIT parameter, see Limiting the use of private memory objects in *z/OS MVS Programming: Extended Addressability Guide*.

Enabling certain TSO/E commands to z/OS UNIX users

To make certain TSO/E commands (such as OEDIT, OBROWSE, OPUTX, OGETX, and ISHELL) and some REXX execs available to users, concatenate the following target libraries to the appropriate ISPF data definition names. The following data sets are for the English panels, messages, and tables:

- SYS1.SBPXPENU concatenated to ISPPLIB.
- SYS1.SBPXMENU concatenated to ISPMLIB.
- SYS1.SBPXTENU concatenated to ISPTLIB.
- SYS1.SBPXEXEC concatenated to SYSEXEC or SYSPROC.

To use the Japanese translation of the panels, messages, and tables, you must concatenate the following target libraries to the appropriate ISPF data definition names.

- SYS1.SBPXPJPN concatenated to ISPPLIB.
- SYS1.SBPXMJPN concatenated to ISPMLIB.
- SYS1.SBPXTJPN concatenated to ISPTLIB.
- SYS1.KHELP concatenated to SYSHELP.

For more information about translation into Japanese, see Chapter 9, “Customizing for your national code page in the shell,” on page 219.

Although the user can invoke these TSO/E commands from a TSO/E command line, most users invoke TSO/E commands or programs from an ISPF menu. For that reason, you should add these TSO/E commands to an ISPF selection panel. In the following example, the ISR@PRIM (ISPF Primary Option Menu) was modified to include these commands.

```
%----------------------  ISPF PRIMARY OPTION MENU ---------
%OPTION  ===>_ZCMD
%
%   0 +ISPF PARMS  - Specify terminal and user parameters
%   1 +BROWSE      - Display source data or output listings
%   2 +EDIT        - Create or change source data
%   3 +UTILITIES   - Perform utility functions
%   4 +FOREGROUND  - Invoke language processors in foreground
%   5 +BATCH       - Submit job for language processing
%   6 +COMMAND     - Enter TSO Command, CLIST, or REXX exec
%   7 +DIALOG TEST - Perform dialog testing
%   8 +LM UTILITIES- Perform library administrator utility functions
%   9 +IBM PRODUCTS- Additional IBM program development products
%   X +EXIT        - Terminate ISPF using log and list defaults

%  1F - Browse files
%  2F - Edit files
% ISH - ISPF Shell

)INIT
  HELP = ISR00003
)PROC
  IF (&ZCMD ¬= ' ')
    &ZQ = TRUNC(&ZCMD,'.')
    IF (&ZQ = ' ')
      .MSG = ISRU000
  &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                0, 'PANEL(ISPOPTA)'
               1F,'CMD(OBROWSE)'
               2F,'CMD(OEDIT)'
              ISH,'CMD(ISHELL)'
              ' ',' '
                *,'?' )
)END
```

You must make the following changes to an ISPF selection panel:

1. Add a statement to the list of options for Browse files. Be sure to include a selection number with the statement. In this example, the statement is:

   ```
   %  1F - Browse files
   ```

2. Add a statement to the)PROC section of the panel to invoke OBROWSE. In this example, the statement is:

   ```
   1F,'CMD(OBROWSE)'
   ```

   Be sure that the symbol at the start of this statement (1F in this example)) matches the number that is specified in the list of options.

3. Add a statement to the list of options for edit files. Include a selection number with the statement. In this example, the statement is:

   ```
   %  2F - Edit files
   ```

4. Add a statement to the)PROC section of the panel to invoke OEDIT. In this example, the statement is:

   ```
   2F,'CMD(OEDIT)'
   ```

   Be sure that the symbol at the start of this statement (2F in this example) matches the number that is specified in the list of options.

5. Add a statement to the list of options for the ISPF shell. Include a selection number with the statement. In this example, the statement is:

   ```
   % ISH - ISPF shell
   ```

6. Add a statement to the)PROC section of the panel to invoke the ISPF shell environment. In this example, the statement is:

   ```
   ISH,'CMD(ISHELL)'
   ```

Be sure that the symbol at the start of this statement (ISH in this example) matches the number that is specified in the list of options.

Tip: If you customize your ISPF TSO Command Table (ISPTCM) to make your default flag differ from the ISPF default of 61, you might have to create new entries in your ISPTCM for some of the TSO/E commands that specify FLAG=61. The OEDIT and OBROWSE commands do not run with some flag values. You can correct this situation by adding ISPTCM entries for BPXWBRWS and BPXWEDIT, restoring the ISPF defaults. If you changed the defaults and do not experience problems with those commands, you should not have to add ISPTCM entries to restore defaults for those commands.

For information about modifying ISPF selection panels, see *z/OS ISPF Planning and Customizing*.

Globalization on z/OS systems

Setting up your system or user environment for globalization on z/OS systems is a little different from what most users are accustomed to when setting up globalization on ASCII platforms. An extra step is typically needed when changing your locale, which involves setting the ASCII/EBCDIC coded character set conversion for the controlling terminal. The conversion is required because most PC terminal emulators require ASCII data, but the z/OS shells use EBCDIC data.

For example, when using a PC emulator to interactively log into an ASCII UNIX operating system, a user will:

- On the PC, change the emulator's coded character set to match the coded character set of the remote session's locale.
- In the UNIX shell, assign the environment variable LC_ALL to a new locale, where the ASCII coded character set of that locale matches the emulator's setting.

When interactively logging into an EBCDIC z/OS UNIX operating system, the user will:

- On the PC, change the emulator's coded character set to match the ASCII coded character set of the remote session's locale. For example, the user might change the translation settings in their emulator to use coded character set ISO/IEC 8859-2 (Latin-2).
- In the UNIX shell:
  - Assign the environment variable LC_ALL to a new locale, whose EBCDIC coded character set is compatible with the ASCII coded character set used in the emulator. To determine if a coded character set is compatible with a particular locale, refer to the information about locales that are supplied with z/OS XL C/C++ in *z/OS XL C/C++ Programming Guide*.

    For example, a user might issue:

    ```
    export LC_ALL=Hu_HU.IBM-1165
    ```

  - If a tty is allocated, issue the `chcp` command to assign the EBCDIC and ASCII coded character sets, as appropriate. Note that the specified ASCII coded character set should match that of the client emulator's setting.

    For example, a user might issue:

    ```
    chcp -a ISO8859-2 -e IBM-1165
    ```

On z/OS systems, in daemons such as rlogind, telnetd, and sshd, conversion between ASCII and EBCDIC occurs in the forked daemon process which handles the user's connection. This process allocates the terminal (tty) for the end user. On ASCII platforms, no conversion is necessary.

Chapter 4. Establishing UNIX security

To provide data and system security, the security administrator and security auditor need to set up and maintain security. z/OS UNIX provides security mechanisms that work with the security offered by the z/OS system. A security product is required, either RACF or an equivalent security product. It is assumed that you are using RACF. If you are using an equivalent security product, you should refer to that product's documentation. If you do not have a security product, you must write SAF exits to simulate all of the functions.

Your installation might need to meet the United States Department of Defense Class C2 criteria specified in *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD. RACF provides the system integrity and user isolation required to meet the requirements for C2-level security.

Additionally, multilevel security functions for z/OS systems build on the work done on MVS to meet the B1 criteria. Using multilevel security, an installation can provide a high level of security on a z/OS system. For more information about multilevel security, refer to *z/OS Planning for Multilevel Security and the Common Criteria*.

When supported by the installed security products, mixed-case passwords and password phrases are also supported by BPX1PWD, BPX1SEC, and BPX1TLS callable services.

Use the following documents as references:

- Activating LDAP change notification in *z/OS Security Server RACF Security Administrator's Guide*
- *z/OS Security Server RACF System Programmer's Guide*
- *z/OS Security Server RACF Command Language Reference*

List of subtasks

Subtasks	Associated procedure
Preparing RACF	“Steps for preparing RACF ” on page 46
Defining z/OS users to RACF	“Steps for defining z/OS UNIX users to RACF” on page 51
Managing group identifiers and user identifiers (GIDs and UIDs)	“Steps for obtaining security information about users” on page 56 “Steps for setting up field-level access ” on page 57
Defining groups to RACF	“Steps for creating z/OS UNIX groups” on page 61
Allowing users to transfer file ownership to any UID or GID on the system	“Steps for authorizing selected users to transfer ownership of any file” on page 66 “Steps for setting up the CHOWN.UNRESTRICTED profile” on page 67
Giving superuser authority to users.	“Steps for setting up BPX.SUPERUSER ” on page 68
Changing superusers from UID(0) to a unique nonzero UID	“Steps for changing a superuser from UID(0) to a unique nonzero UID” on page 69
Setting up the FILE.GROUPOWNER.SETGID profile	“Steps for setting up the FILE.GROUPOWNER.SETGID profile” on page 84

Subtasks	Associated procedure
Creating and activating sanction lists	“Steps for creating a sanction list” on page 96 “Steps for activating the sanction list” on page 97
Maintaining the security level of the system	“Steps for maintaining the security level of the system” on page 99
Activating FSACCESS checking	“Steps for giving selected users or groups access to a z/OS UNIX file system” on page 101

If you require a high level of security in your z/OS system and do not want superusers to have access to z/OS resources such as SYS1.PROCLIB, read the following topics:

- “Comparing UNIX security and z/OS UNIX security” on page 311.
- “Establishing the correct level of security for daemons” on page 313.

Preparing RACF

The security administrator needs to prepare RACF to provide security and to define users to RACF. To be a z/OS UNIX user, the user's default group must be a z/OS UNIX group.

Other security topics include:

- Chapter 17, “Setting up for daemons,” on page 311, for rlogin security considerations
- Chapter 18, “Preparing security for servers,” on page 341, for information about preparing security for servers
- “Steps for preparing the security program for daemons” on page 314

Steps for preparing RACF

Before you begin: You need to have installed a z/OS system and to be aware that a SAMPLIB member, BPXISEC1, is provided with z/OS UNIX. This sample TSO/E CLIST provides all the RACF commands that are needed for the security setup. Use this sample member to set up your security environment.

Perform the following steps to prepare RACF for z/OS UNIX.

1. The OMVS cataloged procedure runs a program that initializes the kernel. Issue ADDGROUP and ADDUSER commands to define the user ID and group ID specified for OMVS. For example:

```
ADDGROUP OMVSGRP OMVS(GID(1))
ADDUSER OMVSKERN DFLTGRP(OMVSGRP)
        OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
        NOPASSWORD
```

   - When you create the RACF user ID for OMVSKERN, use the NOPASSWORD option to create it as a protected user ID. Protected user IDs cannot be used to log on to the system or be revoked by incorrect password or passphrase attempts.
   - Specify the RACF name for the group: OMVSGRP in the example. Because the processes created by `/usr/sbin/init` inherit the GID of the BPXOINIT, do not permit OMVSGRP to any resources, unless programs you start using `/etc/rc` need to be permitted to these resources. For more information, see “Customizing /etc/rc” on page 204.

     In this example, the GID is 1. However, OMVSGRP can have any group ID. The GID assigned to this group should be consistent with the GID assigned to the basic files that are installed in the root directory. ServerPac assumes that the default of GID(1) will be used. If you want to change the GID value, then you must also change all files and directories in the entire z/OS UNIX file system that currently has GID(1) to the new GID value.
   - The TSO/E segment is not needed because NOPASSWORD prevents the OMVSKERN user ID from being used with TSO/E. This prevents a user logon from interfering with the OMVSKERN user ID.

- Assign UID(0) to the kernel user ID (OMVSKERN). Any programs forked by `/etc/rc` receive their authority from the user ID assigned to the BPXOINIT process. Use the same user ID for BPXOINIT as you assigned to the kernel (OMVS). The BPXOINIT process and any programs forked by the kernel's descendants have superuser authority.
- Specify the home directory for the kernel: the root (/).
- To define the default shell for processes run with the OMVSKERN user ID, specify:

```
PROGRAM('/bin/sh')
```

- The initialization process BPXOINIT controls the accounting information for `/usr/sbin/init`, `/etc/rc`, and any other programs it starts. If you want to tailor accounting information for the kernel and startup processes, consider the following:
  - OMVS and BPXOINIT get their account data independently. You can control the account data in the same way that you set up accounting data for any cataloged procedure.
  - The accounting data for `/usr/sbin/init`, `/etc/rc`, and any processes created by `/etc/rc` is obtained from the security product database for user OMVSKERN (the same user ID should be assigned to the BPXOINIT cataloged procedure).
  - The account data for a process started by `/etc/rc` can be set with the _BPX_ACCOUNT environment variable. For example:

```
HOME('/') export _BPX_ACCOUNT=AccountingData
```

---

2. Add the OMVS procedure either to the RACF STARTED class or to the RACF started procedures table, module ICHRIN03. When deciding which method to use, keep in mind that the STARTED class profiles are checked before ICHRIN03, and that any changes made to ICHRIN03 do not take effect until the next IPL. The entry for the OMVS cataloged procedure defines the user ID and group name that the OMVS address space will be assigned.
   - You must decide whether to mark OMVS (the kernel) trusted for access. Making the kernel trusted is useful for giving the kernel access to any local data set that it wants to mount. If you do not mark the kernel trusted for local access, set up profiles so that the kernel user ID has access to any local data set that it needs to mount. For information about trusted attributes, see Assigning the RACF TRUSTED attribute in *z/OS MVS Initialization and Tuning Reference*.
   - Give the entry for the BPXOINIT started procedure the same identity as OMVS. Do not mark BPXOINIT trusted.
   - If you have decided to add OMVS as a trusted procedure, give the kernel the trusted attribute. With the trusted attribute, the kernel can work with the local data sets containing the file systems. Use one of these methods:
     - Add it to the RACF STARTED class:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED OMVS.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP)
TRUSTED(YES))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```

       If you add any other entries after this, you issue `SETROPTS RACLIST(STARTED) REFRESH` and they will be picked up on the next START.

       This defines the BPXOINIT task to run under the user ID OMVSKERN, which should be NOPASSWORD.

     - Add the following entries to ICHRIN03.

```
DC CL8'OMVS'     PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'40'       TRUSTED ATTRIBUTE BIT
DC XL7'00'       RESERVED
```

- If OMVS is not a trusted procedure, add OMVS without making it trusted, using one of the following methods. (See step "5" on page 48 for additional measures needed if the kernel is not trusted.)
  - Add it to the RACF STARTED class:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED OMVS.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP)
TRUSTED(NO))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```

    If you add any other entries after this, issue

```
SETROPTS RACLIST(STARTED) REFRESH
```

    They will be picked up on the next START.
  - Add it to ICHRIN03, as shown in the following example:

```
DC CL8'OMVS'     PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'00'       NOT TRUSTED
DC    XL7'00'       RESERVED
```

---

3. Add the BPXOINIT procedure (it runs the initialization process) without making it trusted, using either one of these methods:
   - Add it to the RACF STARTED class:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED BPXOINIT.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP)
TRUSTED(NO))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```

   - Add it to ICHRIN03:

```
DC CL8'BPXOINIT' PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC    XL1'00'       NOT TRUSTED
DC    XL7'00'       RESERVED
```

---

4. Add the BPXAS procedure without making it trusted. (When programs issue fork or spawn requests, the BPXAS procedure is used to provide a new address space.) Use one of the following methods:
   - Add it to the RACF STARTED class:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED BPXAS.* STDATA(USER(OMVSKERN)
TRUSTED(NO))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```

   - Add it to ICHRIN03:

```
DC CL8'BPXAS' PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC    XL1'00'       NOT TRUSTED
DC    XL7'00'       RESERVED
```

---

5. If you did not make the kernel address space trusted, you must give the kernel access to the local data sets in one of two ways.

   You will need to either fulfill the three following conditions:

- Use consistent qualifiers for the local data set names. For example, use OMVS.*xxxxxxxx*, where OMVS.*xxxxxxxx* is the name for a data set.
- Create a generic RACF profile for the OMVS.* data sets, giving the kernel's user ID (that is, OMVSKERN) ALTER authority. For example:

```
ADDUSER OMVS
ADDSD ('OMVS.*') OWNER(OMVSKERN) UACC(NONE)
PERMIT 'OMVS.*' ACCESS(ALTER) ID(OMVSKERN)
```

- Authorize administrators who will be allocating local data sets by adding their user IDs to the OMVS.* access list in the data set profile and giving them ALTER authority.

or:

- Make sure your administrators who create local data sets give the kernel permission before having the file system mounted. For each local data set, the creator defines a data set profile with UACC(NONE) and gives the kernel address space ALTER authority. For example:

```
ADDUSER SMORG
ADDSD ('SMORG.ZFS')
UACC(NONE) OWNER(SMORG) PERMIT 'SMORG.ZFS'
ACCESS(ALTER) ID(OMVSKERN)
```

If you use the automount options `allocuser` or `allocany`, you must also give the kernel userid UPDATE authority to all catalogs where dynamically allocated automount-managed file systems are cataloged.

---

6. If you are defining colony address spaces for a physical file system (for example, for the NFS client), set up the security by adding an entry to the RACF STARTED class or to the RACF started procedures table for each colony address space. The procedure name specified in the entry must match the ASNAME specified on the FILESYSTYPE statement in the BPXPRMxx member. For example, if you specified the following:

```
FILESYSTYPE  TYPE(...) ENTRYPOINT(...) ASNAME(OMVSCOL1)
```

Then use one of these methods to specify the procedure name:

- Add it to the RACF STARTED class:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED OMVSCOL1.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP)
TRUSTED(NO))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```

- Add the following entry to ICHRIN03, to allow the colony address space to be dubbed as a process with UID(0):

```
DC CL8'OMVSCOL1'       PROCEDURE NAME
DC CL8'OMVSKERN'       USERID
DC CL8'OMVSGRP'        GROUP NAME
DC XL1'00'             NOT TRUSTED
DC XL7'00'             RESERVED
```

When you are done, you have prepared RACF for z/OS UNIX.

Using RACF with z/OS UNIX

The security functions provided by RACF include validating users, file access checking, privileged user checking, and user limit checking. z/OS UNIX users are defined with RACF commands. When a job starts or a user logs on, RACF verifies the user ID and password or password phrase. When an address space requests a z/OS UNIX callable service for the first time, RACF:

1. Verifies that the user is defined as a z/OS UNIX user.
2. Verifies that the user's current connect group is defined as a z/OS UNIX group.
3. Initializes the control blocks needed for subsequent security checks.

For more information about auditing in the RACF environment, see Auditing for z/OS UNIX System Services in *z/OS Security Server RACF Auditor's Guide*.

RACF performance considerations

Associating RACF user IDs and groups to UIDs and GIDs has important performance considerations. Both the UNIXMAP class and virtual lookaside facility (VLF) should remain active and the VLF classes IRRUMAP and IRRGMAP should be defined to VLF. To understand how VLF can affect performance, refer to the following RACF documentation:

- Mapping UIDs to user IDs and GIDs to group names in *z/OS Security Server RACF System Programmer's Guide*.
- Using the UNIXMAP class and Virtual Lookaside Facility (VLF) in *z/OS Security Server RACF Security Administrator's Guide*.

Setting up users and groups

The system provides security by verifying a user and verifying that a user or program can access a process or file. It verifies the user IDs and passwords or password phrases of users when they log on to a TSO/E session or when a job starts. Then it does the following:

- When a user in a TSO/E session invokes the shell: RACF verifies that the interactive users are defined as z/OS UNIX users before the system initializes the shell.
- When a daemon creates a process for a user: RACF verifies that the user is properly defined before the system initializes the process.
- When a program requests a kernel service for the first time: RACF verifies that z/OS UNIX users are running the program before the system provides the service. The types of programs are:
  - Application programs
  - Started procedures
  - Products that use kernel services, such as Resource Measurement Facility (RMF)

Authorize a user to access z/OS UNIX resources by:

- Adding a GID to the RACF group profile for an existing or new RACF group, which is to be defined as the default group of the user
- Adding a UID to the RACF user profile for an existing or new user and connecting each user to a RACF group that has a GID

Activating supplemental groups

When RACF list-of-groups checking is active, a user can access z/OS UNIX resources if they are available to any group the user is connected to and if the group has a GID in its RACF group profile. The additional groups are called *supplemental groups*. To activate the RACF list-of-groups checking, specify the GRPLIST option on the RACF SETROPTS command.

Restrictions:

- The maximum number of supplemental groups that can be associated with a process is 300.
- NFS Client only uses the first 16 groups as supplemental groups when communicating with a remote NFS server.
- A user can be connected to more than 300 groups, but only the first 300 group IDs are identified as the user's supplemental groups. When you issue a LISTUSER command, these are shown as associated with a user's process. It is recommended that all groups be assigned an OMVS GID.

For more information, see Activating list-of-groups checking (GRPLIST option) in *z/OS Security Server RACF Security Administrator's Guide*.

Defining z/OS UNIX users to RACF

You can define z/OS UNIX users to RACF. Alternatively, you can use the ISPF shell to set up existing users with unique UIDs.

Notes:

- You must define the user to your security product as a z/OS UNIX user before you try to make the user's file system available. If you do not, you will get error messages when you try to make it available.
- The OMVS segment also contains the HOME directory and the first PROGRAM that is executed when this user logs into z/OS UNIX or invokes the OMVS TSO/E command. Make sure the HOME directory in the OMVS segment matches the home directory that is defined for that user in the file system.
- The recommended home directory for a user is `/u` followed by the user ID; for example, `/u/user1` would be the home directory for the user1 ID.
- Make sure that unique UIDs are assigned to each user.

  Although you can assign the same UID to multiple users, it is not recommended. However, it may be necessary for some cases, such as superusers. If you assign the same UID to multiple users, control at an individual user level is lost because the UID is used in z/OS UNIX security checks. Users with the same UID assignment are treated as a single user during z/OS UNIX security checks.

Restriction: The limit on the number of user IDs that can share a UID when the RACF database is using AIM is 129.

Steps for defining z/OS UNIX users to RACF

About this task

This task explains the steps for defining z/OS UNIX to RACF.

Before you begin: You need to log on the user ID with RACF SPECIAL authority.

Perform the following steps to define z/OS UNIX users to RACF.

Procedure

1. Authorize a user to z/OS UNIX by entering:
   - A RACF ADDUSER command for each new user to be given access to z/OS UNIX resources. The ADDUSER command creates a RACF user profile.
   - A RACF ALTUSER command for each current user who is to be given access to z/OS UNIX resources. The ALTUSER command changes a current RACF user profile.

   To provide access to z/OS UNIX resources, both ADDUSER and ALTUSER have an OMVS parameter. The UID subparameter specifies the UID, while the AUTOUID subparameter specifies that RACF is to assign an unused UID value.

   ———————————————————————————————————————
2. Assign a home directory for each user through the HOME subparameter on the ADDUSER or ALTUSER command.

   Example: If the home directory is `/u/john`, specify:

   ```
   HOME('/u/john')
   ```

   The home directory should be fully qualified (`'/u/john'`). If a home directory is partially specified (for example, `john`) problems might during process initialization. Then, create that home directory for

each user. The home directory, like all file names, is case-sensitive. It is recommended that the user name in the home directory be entered in lowercase.

Alternatively, you can use the ISPF shell to define a home directory for each user.

Example: If the home directory is the root, specify:

```
HOME('/')
```

In similar open systems, the directory that is used for users is /u and the name of the user's home directory is the user name that is associated with the user. In a z/OS system, the user name is the user ID:

- If a user accesses the shell from TSO/E, the user ID is folded to uppercase.
- With rlogin, the user ID is case-sensitive. If the alias table (USERIDALIASTABLE) is not set up, then case does not matter and the user ID is folded. If the alias table is being used and the user ID is found in it, then the case-sensitive user ID for UNIX activity is used.

_____

3. Specify an initial program for each user through the PROGRAM subparameter of the ADDUSER or ALTUSER command.

   ```
   PROGRAM('/bin/sh')
   ```

   Alternatively, you can use the ISPF shell to specify an initial program for each user.

   The system gives control to the user program when the user logs in or invokes the OMVS command. The PROGRAM value is also used for the `rlogin`, `otelnetd`, `su`, and `newgrp` commands, where a shell is to be created.

   _____

4. Do one of the following tasks to connect a user to an already-defined RACF group. The RACF group must have an OMVS GID for the user to access z/OS UNIX resources.

   - Specify the RACF group on the DFLTGRP parameter on the RACF ADDUSER command. The specified group becomes the user's default group.

     If you do not specify a RACF group on the RACF ADDUSER command, your current group becomes the user's default group.
   - Enter a RACF CONNECT command to connect a user to the RACF group. Specify the DFLTGP parameter on the RACF ALTUSER command to change the user's default group to the RACF group with an OMVS GID.

     To use z/OS UNIX resources, the default group of the user must have a GID defined.

   _____

5. z/OS UNIX performs SYSOUT tailoring for every forked address space. When defining the users, code the WORKATTR parameter to specify the user's name and address. The name and address appear on the user's SYSOUT output.

   _____

Results

When you are done, you have defined z/OS UNIX users to RACF.

In similar open systems, the `/etc/passwd` file contains definitions for the HOME, SHELL, and LOGNAME environment variables. z/OS UNIX provides better security by keeping these values in the RACF user profile.

Example: The following example shows an ADDUSER command to create a new user ID, JOHN, with authority to use z/OS UNIX.

```
ADDUSER JOHN DFLTGRP(ENGNGP7) NAME('JOHN DOE') PASSWORD(A4B3C2D1)
        OMVS(UID(314) HOME('/u/john') PROGRAM('/bin/sh'))
        TSO(ACCTNUM(12345678) DEST(P382005) PROC(PROC01) SYSOUTCLASS(A))
        WORKATTR(WANAME('JOHN DOE') WAACCNT(12345678)
        WABLDG(507_PARK_PLACE) WAROOM(124)
        WADEPT(ENGNG555) WAADDR1(WIDGET_INC) WAADDR2(NEW_YORK)
        WAADDR3(NEW_YORK) WAADDR4(10002))
```

The DFLTGRP parameter places user ID JOHN in the RACF group ENGNGP7, which has a GID of 678. The OMVS parameter on the ADDUSER command takes the following actions:

- Gives JOHN a UID of 314.
- Invokes the shell in the file `/bin/sh` when John Doe enters a TSO/E OMVS command.
- Gives JOHN a home directory of `/u/john`. The home directory needs to be added to the file system.

  On an open system, a working directory is normally defined in lowercase letters and typically has the user's user ID as its name—for example, `/u/john`. If a REXX exec or CLIST extracts the user ID with a *&userid* variable, the value that is returned is in uppercase: JOHN. If the REXX exec or CLIST appends the returned value to `/u`, the result is `/u/JOHN`. `/u/john` and `/u/JOHN` are two different directories. Consider this behavior in using REXX execs, CLISTs, C programs, or programs using the callable services where the functions return user IDs.
- Specifying the WORKATTR for user ID JOHN allows daemons to create processes with the correct accounting and SYSOUT defaults. For example, if JOHN logs into the system by using a rlogin command from a workstation, a new process is created for JOHN using the attributes from the WORKATTR.

Storing user-specific information in OMVS segments

Information specific to the user is kept in the OMVS segment of the user profile, while information specific to groups is kept in the OMVS segment of the group profile.

- The OMVS segment of the user profile contains information such as the user identifier (UID) and the path name of the HOME directory.
- For the group profile, the OMVS segment contains the group identifier (GID).

Before users and groups can request access to z/OS UNIX services, the OMVS segments of the profiles associated with them must be defined.

Every address space that is dubbed must have a security environment with a valid OMVS segment at the address space level. In a multiuser address space, if a task is dubbed and it has a security environment that is different than the address space, then the user ID identified with the task must also have a valid OMVS segment.

Automatically generating OMVS segments

Unique UIDs and GIDs can be generated on demand for users and groups that do not have OMVS segments defined. RACF saves the generated UIDs and GIDs in new OMVS segments that are created for user and group profiles in the RACF database. For more information. see automatic UID and GID generation in *z/OS Security Server RACF Security Administrator's Guide*.

Tip: RACF allows a string that contains `&racuid` to include the user ID of the user in the home directory string. If that is used, then home directories need to be created. For more information about creating home directories, see Steps for automatically assigning unique IDs through UNIX services in *z/OS Security Server RACF Security Administrator's Guide*.

Any z/OS UNIX command or callable service that can specify an MVS user ID can also cause the automatic creation of OMVS segments if that user ID does not have currently have any OMVS segments.

Before the OMVS segment can be created, the BPX.UNIQUE.USER profile must be defined in the FACILITY class. A model user can be specified in the APPLDATA field for BPX.UNIQUE.USER. The OMVS segment

from the model user is used to initialize new OMVS segments for the user profile; this includes all attributes (HOME, PROGRAM, and user limits) except the UID.

The BPX.NEXT.USER profile in the FACILITY class is used by RACF to derive unused UID and GID values. The FACILITY class does not have to be active for RACF to use BPX.NEXT.USER. When creating the BPX.NEXT.USER profile, generic characters cannot be used in the name. The APPLDATA field for BPX.NEXT.USER can specify either a starting UID or GID value or range of values for generating unique UIDs and GIDs. After RACF determines the next unique UID, the UID is saved in the newly created OMVS segment for the user profile. Similar processing is done when a starting GID or range of values is specified; if the group associated does not have a GID, the GID is saved in a newly created OMVS segment for the group profile.

Figure 6 on page 54 illustrates how the unique UID assignment process derives the UID and GID values from the BPX.NEXT.USER profile and saves the values in the OMVS segment for the user profile (MYUSER) and the OMVS segment for the group profile (MYGROUP). The figure also shows how a user profile indicated in the BPX.UNIQUE.USER profile can be the source of other OMVS information that is copied to the user profile (MYUSER).

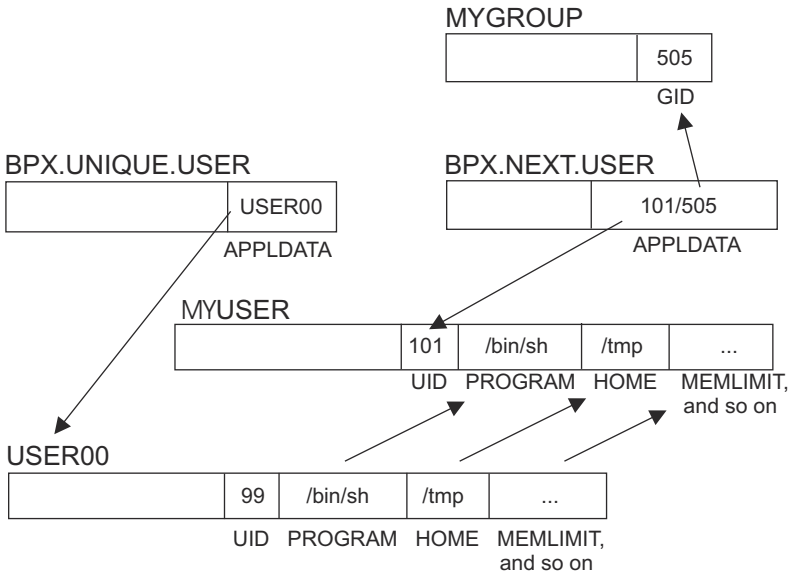


Figure 6. How unique UIDs and GIDs are assigned

For the requirements to enable automatic UID and GID assignment, see automatic assignment in *z/OS Security Server RACF Security Administrator's Guide*.

You can specify the RACF string `&RACUID` as a placeholder for the user ID in the home directory path name. When RACF creates the OMVS segment, it substitutes the user ID for which the OMVS segment is being created. When automount is implemented, a user file system is allocated, mounted, and assigned the user ID as its owner. For more information about specifying `&racuid` and considerations for sharing the RACF database, Steps for automatically assigning unique IDs through UNIX services in *z/OS Security Server RACF Security Administrator's Guide*.

Security implications

Executable programs are generally categorized as coming from authorized or unauthorized libraries. Programs in authorized libraries are considered safe for anyone to run. That is, the code should be free of viruses and should uphold the integrity and security classification of the operating system.

Programs in unauthorized libraries can be further divided into system-controlled libraries, which are protected from general user modification, and libraries that are not system-controlled. Libraries that are not system controlled are not considered safe for anyone to run. This code is generally a local version of a program that the owner has created or modified. Users with special privileges, must use caution when running such programs. If a programmer with RACF SPECIAL or authority to update APF-authorized

libraries runs a program from an uncontrolled library, it is possible for the program to take advantage of the caller's authority to compromise the integrity of the system.

The BPX.DEBUG resource in the FACILITY class enables you to debug APF-authorized programs, using ptrace via dbx. For more information about BPX.DEBUG, see "Setting up the UNIX-related FACILITY and SURROGAT class profiles" on page 72.

There are additional considerations when combining traditional MVS services and z/OS UNIX.

The entire file system is considered to be an unauthorized library. You can authorize individual programs within the file system as APF-authorized by setting the APF-extended attribute. Programs that are APF-authorized behave the same as other programs that are loaded from APF-authorized libraries. If a program running in an APF-authorized address space attempts to load a program from the file system that does not have the APF-extended attribute set, the load is rejected. This applies to non-jobstep exec, local spawn, attach_exec, and DLL loads. This is consistent with the way that Contents Supervisor rejects requests to LINK, LOAD, or ATTACH unauthorized programs from an authorized environment.

In order to run a program from the file system in an APF-authorized address space, you have two choices:

1. You can link-edit the program into an APF-authorized library and turn on the sticky bit, using the `chmod` command.
2. You can use the `extattr` command to set the APF-authorized extended attribute of the file.

If an APF-authorized program is the first program to be executed in an address space, then you also need to set the authorization code to 1 (AC=1) when your program is link-edited. If a program is loaded into an APF-authorized address space but is not the first program to be executed, it should not have the AC=1 attribute set.

Checking user and group authority

The system uses two types of user and group IDs to check a user's authority to access different RACF-protected resources. Examples of two RACF-protected resources are MVS data sets and local files.

MVS data sets
The system uses:

- The user's RACF profile
- The RACF group name for the user's current group
- The RACF group name for each group the user is connected to, if list-of-group checking is active

Local files
The system uses:

- The effective UID
- The effective GID
- The GIDs for the supplemental groups, if list-of-group checking is active

Users must have a UID and GID defined when entering the TSO/E OMVS command and for certain kernel services.

Users also need search authority to all directories in the path name for their home directory. Set these permissions for each directory using the `chmod` command and either the MODE operand of the TSO/E MKDIR command or the *mode* option of the `mkdir` command that creates a directory. For more information, see "Controlling access to files and directories" on page 83.

Obtaining security information about groups

Use the RACF LISTGRP command to list details of specific RACF group profiles. A group profile consists of a RACF segment and, optionally, other segments such as DFP and OMVS. The LISTGRP command provides you with the option of listing the information contained in the entire group profile (all segments), or listing the information contained only in a specific segment of the group profile.

Steps for obtaining security information about a group

Before you begin, you need to have a user ID that has the needed RACF authority. For the RACF authority you need, see LISTGRP in *z/OS Security Server RACF Command Language Reference*.

Perform the following steps to check the OMVS security information for a group.

1. Log on to your user ID.

   _____

2. Issue a RACF LISTGRP command with the OMVS operand and the RACF group name.

   For example, to list the GID information for the RACF group ENGNP7:

   ```
   LISTGRP ENGNGP7 OMVS NORACF
   ```

   _____

You should now see information from the RACF group profile. If the RACF group was assigned a GID, the profile identifies the GID. All groups that OMVS users belong to should have OMVS GIDs.

Obtaining security information about users

You can obtain security information for users if the security administrator has set up field-level access for users for the OMVS segment of the RACF user profile.

Steps for obtaining security information about users

Before you begin, you need to have a user ID that has the needed RACF authority. For the RACF authority you need, see LISTUSER (List user profile) in *z/OS Security Server RACF Command Language Reference*.

Perform the following steps to check the OMVS segment of the security information for a user.

1. Log on to your user ID.

   _____

2. Issue a RACF LISTUSER command with the OMVS operand and the user ID.

   For example, to list the OMVS information for user ID JOHN:

   ```
   LISTUSER JOHN OMVS NORACF
   ```

   _____

You should now see lists from the user's RACF user profile the fields the user has authority to see. The fields can be:

- The OMVS UID.
- The user's home directory.
- The program (typically the shell, called when the user invokes it by using the TSO/E OMVS command, rlogin, or telnet).
- The user limits.

Setting up field-level access for the OMVS segment of a user profile

To allow a user to see or change OMVS fields in a RACF user profile, you can set up field-level access. You can authorize a user to specified fields in any profile or to specified fields in the user's own profile. To authorize users to the OMVS fields in their own profiles, use the ISPF shell, or issue the RACF and PERMIT commands as described in “Steps for setting up field-level access ” on page 57.

Steps for setting up field-level access

Before you begin, you need to know which users need to have field-level access.

Perform the following steps to set up field-level access for the OMVS segment of a user profile.

1. Define a profile for each of the OMVS fields with a RACF RDEFINE command. For example:

```
RDEFINE FIELD USER.OMVS.UID       UACC(NONE)
RDEFINE FIELD USER.OMVS.HOME      UACC(NONE)
RDEFINE FIELD USER.OMVS.PROGRAM   UACC(NONE)
RDEFINE FIELD USER.OMVS.CPUTIME   UACC(NONE)
RDEFINE FIELD USER.OMVS.ASSIZE    UACC(NONE
RDEFINE FIELD USER.OMVS.FILEPROC  UACC(NONE)
RDEFINE FIELD USER.OMVS.PROCUSER  UACC(NONE)
RDEFINE FIELD USER.OMVS.THREADS   UACC(NONE)
RDEFINE FIELD USER.OMVS.MMAPAREA  UACC(NONE)
RDEFINE FIELD USER.OMVS.MEMLIMIT  UACC(NONE)
RDEFINE FIELD USER.OMVS.SHMEMMAX  UACC(NONE)
```

_____

2. Permit users to access the fields with RACF PERMIT commands.

   The following example shows commands for the three fields.

   - &RACUID allows all users to look at their own fields.
   - READ access allows users to read the UID field.
   - UPDATE access allows users to change their home directory in the HOME field or the program invoked for a TSO/E OMVS command in the PROGRAM field.

   Give only selected users update access to the UID field and the user limits field. Users with UPDATE access can become a superuser by changing the UID to 0.

```
PERMIT USER.OMVS.UID     CLASS(FIELD) ID(&RACUID) ACCESS(READ)
PERMIT USER.OMVS.HOME    CLASS(FIELD) ID(&RACUID) ACCESS(UPDATE)
PERMIT USER.OMVS.PROGRAM CLASS(FIELD) ID(&RACUID) ACCESS(UPDATE)
```

_____

3. Activate the FIELD class with the RACF SETROPTS command. For example:

```
SETROPTS CLASSACT(FIELD) RACLIST(FIELD)
```

_____

When you are done, you have set up field level access.

For the other parameters on the RDEFINE, PERMIT, and SETROPTS commands, see *z/OS Security Server RACF Command Language Reference*.

Defining group identifiers (GIDs)

You can assign a group identifier (GID) to a RACF group by specifying a GID value in the OMVS segment of the RACF group profile or by using the AUTOGID keyword. When a GID is assigned to a group, all users connected to that group who have a user identifier (UID) in their user profile and whose default or current connect group has a GID in the group profile can use z/OS UNIX functions and can access z/OS UNIX files based on the GID and UID values assigned.

Restriction: The limit on the number of groups that can share a GID when the RACF database is using AIM is 129.

Do not assign the same GID to multiple RACF groups. If you do,control at an individual group level is lost because the GID is used in z/OS UNIX security checks. RACF groups that have the same GID assignment are treated as a single group during z/OS UNIX security checks. They must use the SHARED keyword of the RACF ADDGROUP or ALTGROUP command if the SHARED.IDS profile is defined in the UNIXPRIV

class. For more information about SHARED.IDS, see SHARED.IDS in *z/OS Security Server RACF Security Administrator's Guide*.

If you are using NFS, see "Assigning UIDs and GIDs in an NFS network" on page 60 for more information.

For information about using the RACF list-of-groups checking (GRPLIST) option for access to the files and directories in the z/OS UNIX file system, see special considerations in *z/OS Security Server RACF Security Administrator's Guide*.

Defining user identifiers (UIDs)

The limit on the number of user IDs that can share a UID when the RACF database is using AIM is 129.

Assigning UIDs to single users

You can assign a z/OS UNIX user identifier (UID) to a RACF user by specifying a UID value in the OMVS segment of the RACF user profile or by using the AUTOUID keyword.

When assigning a UID to a user, make sure that the user is connected to at least one group that has an assigned GID. This group must be either the user's default group or one that the user specifies during logon or on the batch job. A user with a UID and a current connect group with a GID can use z/OS UNIX functions and access z/OS UNIX files based on the assigned UID and GID values. If a UID and a GID are not available as described, the user cannot use z/OS UNIX functions.

If you are using NFS, see "Assigning UIDs and GIDs in an NFS network" on page 60 for more information.

Assigning UIDs to multiple users

Do not assign the same UID to multiple user IDs because the sharing of UIDs allows each user to access all of the resources associated with the other users of that shared user ID. The shared access includes not only z/OS UNIX resources such as files, but also includes the possibility that one user could access z/OS resources of the other user that are normally considered to be outside the scope of z/OS UNIX.

However, you might want to assign the same UID to multiple user IDs if these user IDs are used by the same person or persons. It might also be necessary to assign multiple users a UID of 0 (superuser authority). When doing this, it is important to remember that a superuser is implicitly a trusted user who has the potential of using UID(0) to access all z/OS resources.

Important: If the SHARED.IDS profile is defined in the UNIXPRIV class, in order to assign a UID that is already in use to another user ID you must specify the SHARED keyword with the UID keyword on the RACF ADDUSER or ALTUSER command.

By default, RACF does not prevent the sharing of UIDs and GIDs. However, you can enforce unique UNIX identifiers by defining a profile called SHARED.IDS in the UNIXPRIV class. For more information, see SHARED.IDS in *z/OS Security Server RACF Security Administrator's Guide*.

Setting limits for users

You can control the amount of resources consumed by certain z/OS UNIX users by setting individual limits for these users. The resource limits for the majority of z/OS UNIX users are specified in the BPXPRMxx parmlib member. Instead of assigning superuser authority to application servers and other users so they can exceed BPXPRMxx limits, you can individually set higher limits to these users, as discussed in "System limits and process limits " on page 364. Setting user limits allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

Specify limits for z/OS UNIX users by choosing options on the ADDUSER or ALTUSER commands. The limits are stored in the OMVS segment of the user profile. You can set the following limits in the OMVS user segment:

ASSIZEMAX
Maximum address space size (RLIMIT_AS)

CPUTIMEMAX
Maximum CPU time (RLIMIT_CPU)

FILEPROCMAX
Maximum number of concurrently open files per process

MEMLIMIT
Maximum size of storage above the bar

MMAPAREAMAX
Maximum memory map size

PROCUSERMAX
Maximum number of processes for this UID

SHMEMMAX
Maximum size of shared memory

THREADSMAX
Maximum number of threads per process

After you set individual user limits for users who require higher resource limits, you should consider removing their superuser authority, if they have any. You should also reevaluate your installation's BPXPRM*xx* limits and consider reducing these limits. See "Customizing the BPXPRMxx member of SYS1.PARMLIB" on page 17 for more information.

Defining protected user IDs

You can define protected user IDs for started procedures associated with z/OS UNIX, such as the kernel, the initialization started procedure, and daemons that are critical to the availability of your z/OS UNIX system. Defining the protected user IDs will prevent these user IDs from being revoked through inadvertent or malicious incorrect password or password phrase attempts, or from being used for other purposes, such as logging on to the system. For more information, see protected user IDs in *z/OS Security Server RACF Security Administrator's Guide*.

Defining the terminal group name

Certain shell commands, such as `mesg`, `talk`, and `write` require pseudoterminals to have a group name of TTY. When a user logs in, or issues the OMVS command from TSO/E, the group name associated with these terminals is changed to TTY. As part of installation, you had to define the group TTY or use the group alias support as described in "Security requirements for ServerPac and CBPDO installation" on page 80.

Give this group a unique GID and do not connect users to this group.

To make it easier to transport the data sets from test systems to production systems, check that this entry is duplicated in all of your security data bases, including the same UID and GID values in the OMVS segment.

Managing user and group assignments

To prevent duplication, only one or two administrators should assign UIDs and GIDs. To manage UID and GID assignments, use one of the following methods:

- Use the AUTOGID or AUTOUID keywords to have UIDs and GIDs automatically assigned to the user. These keywords on RACF commands define users and groups. This method is suggested.
- Use the RACF database unload utility to move RACF data into a Db2® database and then use the Structured Query Language (SQL) to query the database.
- Use the ISPF shell to perform the tasks of defining users and groups.

Assigning UIDs and GIDs in an NFS network

Network File System (NFS) enables users to mount file systems from other systems so that the files appear to be locally mounted. You end up with a mixture of file systems that come from systems where the UIDs and GIDs might be independently managed. To maintain good security on your local files in an NFS network, the system programmer or the UNIX system programmer must coordinate the UIDs and GIDs on all of the systems. For example, you don't want user RALPH to have UID(7) on system 1 and user SMORG to have UID(7) on system 2. If you use NFS to mount a file system from system 2 on system 1, then user RALPH can access any of user SMORG's files because they both have UID(7).

Assigning identifiers for users

Assigning the same UID to more than one person is strongly discouraged. If you assign the same UID to more than one user ID, z/OS UNIX and RACF treat, in some ways, the users as if they were a single z/OS UNIX user. For example:

- The users share the same MAXPROCUSER limit, which is defined in the BPXPRMxx member, unless each user profile contains its own user limit for MAXPROCUSER.
- The users count as a single user for the MAXUIDS limit in BPXPRMxx.
- One user can enter the `kill` command for the other's processes.
- The users share ownership and access to the same files.
- Services such as the getpwuid() callable service cannot distinguish which user is meant. Such services return data about one of the users, but which user is unpredictable.

If you assign users the same UID, you should warn them of the effects. For UID(0), the effects are less significant because superusers have access to all processes and files and because most BPXPRMxx limits are not enforced against superusers.

To assign a non-unique UID, you can use the SHARED keyword of the RACF ADDUSER or ALTUSER command if the SHARED.IDS profile is defined in the UNIXPRIV class.

Assigning identifiers for groups

All groups should be assigned unique GIDs. If you assign groups the same GIDs, you should warn users of the following effects:

- The groups share ownership and access to the same files.
- Security audit records show the GID, but do not show the RACF group if it was in the supplemental group list; see “Activating supplemental groups” on page 50.
- Services such as the getgrgid() callable service cannot distinguish which group is meant. The services return data about one of the groups, but which group is unpredictable.

To assign a non-unique GID, you can use the SHARED keyword of the RACF ADDGROUP or ALTGROUP command if the SHARED.IDS profile is defined in the UNIXPRIV class.

Upper limits for GIDs and UIDs

RACF allows for UIDs and GIDs within the range of 0-2,147,483,647. However, the `tar` command and some interchange formats supported by the `pax` command might not be able to properly handle values above 2,097,151. Because they are used often, you should take these limitations into consideration when assigning UIDs and GIDs.

When using `pax` or `tar`, UIDs and GIDs greater than 2,097,151 will not be restored correctly unless one of the following conditions are met:

- The archive is created using the `pax` format with the `pax` command.
- The archive is created using the USTAR format and the user or group name associated with the UID or GID exists on the target system. However, an incorrect UID or GID might be restored. The reason that might happen is described next.

When restoring a UID or GID, if the pax or USTAR format was used during writing, pax and `tar` will first attempt to set the UID or GID during the restore using the user or group name stored in the archive. (Of course, the user must have the appropriate privileges to set the UID or GID). If this name is defined on the target system, then the UID or GID is set to whatever UID or GID is associated with the name defined on the target system. (The UID or GID is set, whether or not it matches the UID or GID in the archive, which means that this could be a problem if the name stored on the target system is coincidental rather than intentional).

If the user or group name is not defined on the target system, or if the archive is using the original tar format, then the UID or GID stored in the archive is used. If the UID or GID was originally greater than 2,097,151, and the archive was not created with the pax format, then the archive contains an incorrect version of the UID or GID value due to truncation. This situation will then result in that same incorrect UID or GID value being restored. However, if the archive was created with the pax format, then the original correct UID or GID is restored. The correct values are restored because the pax format supports UID or GID values up to 2147 483647.

In summary, large UIDs and GIDs might not be correctly restored by pax and `tar`. Using the pax format (the preferred method) can avoid this situation, because it supports values up to 2,147,483,647. (the maximum supported by RACF). Using the USTAR format might also avoid this situation, but only if the target system has the same user or group name defined and that name represents the same user or group as it did on the source system. Or, use UID or GID values within the limits for the archive format being used.

For more information about the **pax** and **`tar`** commands, see:

- pax - Interchange portable archives in *z/OS UNIX System Services Command Reference*
- tar: Manipulate the tar archive files to copy or back up a file in *z/OS UNIX System Services Command Reference*

Creating z/OS UNIX groups

A user must belong to at least one group and can be connected to additional groups. When a user connects to the system (that is, logs on to a TSO/E session), one of the groups is selected as the user's current group. For a user to be able to request kernel services and invoke the shell, the user's current RACF group must have a group ID (GID) assigned to it. All groups that a user belongs to should be assigned an OMVS GID. Also, the user's default group must have a GID assigned for POSIX standards conformance.

Steps for creating z/OS UNIX groups

Before you begin, you need to know which RACF group profiles will be used as z/OS UNIX groups.

Perform the following steps to define RACF groups that can be used as z/OS UNIX groups.

1. Log on to the user ID with RACF SPECIAL authority.

   _____

2. Issue one of the following commands. Base your choice on your particular situation.

This table shows the tasks for creating z/OS UNIX groups.

If you want to . . .	Then issue. . .
Define a new RACF group profile and have it be used as a z/OS UNIX group	The ADDGROUP command. To define a RACF group profile named SYS1 and to give it a GID of 575, issue: `ADDGROUP OMVSGRP SUPGROUP(SYS1)` `  OWNER(SYS1) OMVS(GID(575))` This command defines a RACF group profile and created a z/OS UNIX group.
Change a current RACF group profile and have it used as a z/OS UNIX group	The ALTGROUP command. To add a GID of 678 to the current RACF group ENGNGP7, issue: `ALTGROUP ENGNGP7 OMVS(GID(678))` This command creates a z/OS UNIX group. Use AUTOGID to automatically assign an unused GID. For example: `ALTGROUP ENGNGP7 OMVS(AUTOGID)` To assign OMVS GIDs to all groups, use the ISPF shell.

For useful reports and auditing, assign a unique GID to each RACF group name. Reports for the RACF group name will then supply information about the corresponding GID.

When you are done, you have created a z/OS UNIX group. When the user connects to the system (for example, logs on to a TSO/E session), one group is selected as the user's current group. When a user becomes a z/OS UNIX user, the GID of the user's current group becomes the effective GID of the user's process. The user can access resources available to members of the user's effective GID.

Superusers in z/OS UNIX

Your installation defines certain system programmers, users, and started procedures as *superusers*. Superusers pass all security checks and can access any file in the file system. They can do administrative activities such as the following:

- Change the contents of any file.
- Install products.
- Manage processes.
- Change identity from one UID to another.
- Use setrlimit() or prlimit to increase any of the system limits for a process.

Superusers can also have an unlimited number of processes that are running concurrently. For a started procedure, this is true only if it has a UID of 0. It is not true for a trusted or privileged process with a different UID.

When not doing activities that require superuser authority, the superuser joins the majority of users or programs with user authority, which permits access to their own files and certain files to which they have access, according to permission bits.

Rule: The user ID associated with a started procedure needing superuser authority must have a UID, but the UID can have any value. Users running with the trusted or privileged attribute are considered superusers even if their assigned UID is a value other than 0.

The parent process propagates its UID and TRUSTED or PRIVILEGED attribute to a forked child process. Thus, a UID of 0 is propagated to a forked child.

As you are defining users, you might want to define some of them with appropriate superuser privileges. There are three ways to assign superuser authority.

- With the UNIXPRIV class profiles, the preferred way. See “Using UNIXPRIV class profiles ” on page 63.
- With the BPX.SUPERUSER resource in the FACILITY class. See “Using the BPX.SUPERUSER resource in the FACILITY class” on page 68.
- Assigning a UID of 0, which is the least desirable way. See “Assigning a UID of 0” on page 72.

  For specific installation requirements regarding superuser authority, see “Security requirements for ServerPac and CBPDO installation” on page 80.

While some functions require a UID of 0, in most cases you can choose among the three ways. When choosing among them, try to minimize the number of "human" user IDs (as opposed to started procedures) set up with UID(0) superuser authority. However, in z/OS, RACF allows certain users to perform specific privileged functions without being defined as UID(0). BPX.SUPERUSER allows you to request that you be given such access, but you do not have the access unless you make the request. And, the UNIXPRIV class allows you to do other privileged functions, such as mounting a file system. Both these definitions are similar to having UID(0) in that, before RACF grants access to a system resource or use of it, the system checks these definitions.

Do not confuse superuser authority with MVS supervisor state. Being a superuser is not related to supervisor state, PSW key 0, and using APF-authorized instructions, macros, and callable services.

Using UNIXPRIV class profiles

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. By defining profiles in the UNIXPRIV class, you can specifically grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. This way, you can minimize the number of assignments of superuser authority at your installation and reduces your security risk.

Resource names in the UNIXPRIV class are associated with z/OS UNIX privileges. You must define profiles in the UNIXPRIV class protecting these resources in order to use RACF authorization to grant z/OS UNIX privileges. The UNIXPRIV class must be active and SETROPTS RACLIST must be in effect for the UNIXPRIV class. Global access checking is not used for authorization checking to UNIXPRIV resources.

Table 7 on page 63 shows each resource name available in the UNIXPRIV class, the z/OS UNIX privilege that is associated with each resource, and the level of access that is required to grant the privilege.

Table 7. Resource names in the UNIXPRIV class for z/OS UNIX privileges

Resource name	**z/OS UNIX privilege and required minimum access**
CHOWN.UNRESTRICTED	Allows users to use the chown command to transfer ownership of their own files. No minimum access is required. See “Steps for setting up the CHOWN.UNRESTRICTED profile” on page 67.

Table 7. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)	
Resource name	**z/OS UNIX privilege and required minimum access**
CONTAINERS	Allows the user to perform container-related functions. Functions include: • Assigning values to the _BPX_CONTAINER_POD_ID, _BPX_CONTAINER_ID, and _BPX_CONTAINER_QUAL environment variables. • Creating namespaces and altering process namespace affiliations using the clone, unshare, and setns callable services. • Issuing bind mounts and mounting union file systems within a container from a program-controlled address space. The minimum required access is READ.
RESTRICTED.FILESYS.ACCESS	Specifies that RESTRICTED users cannot gain file access by virtue of the `other` permission bits. To override it for a specific user or group, the required minimum required access is READ.
SHARED.IDS	Allows users to assign UID and GID values that are not unique. The minimum required access is READ.
SUPERUSER.FILESYS.ACLOVERRIDE	Specifies that ACL contents override the access that was granted by SUPERUSER.FILESYS. No minimum access is required. It can be overridden for specific users or groups. The user or group must have the same access that would be required to SUPERUSER.FILESYS while accessing the file.
SUPERUSER.FILESYS	To allow the user to read any local file, and to read or search any local directory, the minimum required access is READ. To allow the user to write to any local file, and includes privileges of READ access, the minimum required access is UPDATE. To allow the user to write to any local directory, and includes privileges of UPDATE access, the minimum required access is CONTROL or higher. Authorization to the SUPERUSER.FILESYS resource provides privileges to access only local files. No authorization to access Network File System (NFS) files is provided by access to this resource. READ, UPDATE, and CONTROL (or higher) does not grant permission to update extended attributes of files. This is not equivalent to being a superuser.
SUPERUSER.FILESYS.CHANGEPERMS	Allows users to use the `chmod` command to change the permission bits of any file and to use the `setfacl` command to manage access control lists for any file. The minimum required access is READ.
SUPERUSER.FILESYS.CHOWN	Allows users to use the `chown` command to change ownership of any file. The required minimum access is READ.
SUPERUSER.FILESYS.DIRSRCH	Allows users to read and search any local directories. The required minimum access is READ.

Table 7. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)

Resource name	z/OS UNIX privilege and required minimum access
SUPERUSER.FILESYS.MOUNT	• With the nosetuid option, allows user to issue the TSO/E MOUNT command or the **mount** shell command. Also allows users to unmount a file system with the TSO/E UNMOUNT command or the **unmount** shell command that is mounted with the nosetuid option. Users who are permitted to this profile can use the **chmount** shell command to change the mount attributes of a specified file system. The minimum required access is READ. • With the setuid option, allows user to issue the TSO/E MOUNT command or the **mount** shell command with the setuid option. Also allows user to issue the TSO/E UNMOUNT command or the **unmount** shell command. Users who are permitted to this profile can issue the **chmount** shell command on a file system that is mounted with the setuid option. The minimum required access is UPDATE.
SUPERUSER.FILESYS.QUIESCE	To allow the user to issue `quiesce` and `unquiesce` commands for a file system that is mounted with the nosetuid option, the minimum required access is READ. To allow the user to issue `quiesce` and `unquiesce` commands for a file system that is mounted with the setuid option, the minimum required access is UPDATE.
SUPERUSER.FILESYS.PFSCTL	For more information about the pfsctl callable service, see pfsctl (BPX1PCT, BPX4PCT) — Physical file system control in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. For information about the zFS-specific pfsctl functions, see pfsctl (BPX1PCT) in *z/OS File System Administration*. For detailed information about the use of pfsctl, see Using pfsctl (BPX1PCT) physical file system for z/OS UNIX in *z/OS DFSMSdfp Advanced Services*.
SUPERUSER.FILESYS.USERMOUNT	• With the nosetuid option, allows nonprivileged users to mount and unmount file systems. The minimum required access is READ. • With the setuid option, allows nonprivileged users to mount and unmount file systems. The minimum required access is UPDATE.
SUPERUSER.FILESYS.VREGISTER	Allows a server to use the vreg() callable service to register as a VFS file server. The minimum required access is READ. The SUPERUSER.FILESYS.VREGISTER resource only allows a server such as NFS initialization. Users who are connected as clients through facilities such as NFS do not get special privileges based on this resource or other resources in the UNIXPRIV class.
SUPERUSER.IPC.RMID	Allows user to issue the `ipcrm` command to release any IPC resources. The minimum required access is READ.

Table 7. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)	
Resource name	**z/OS UNIX privilege and required minimum access**
SUPERUSER.PROCESS.GETPSENT	Allows user to use the w_getpsent() callable service to receive data for any process. Also allows users of the **ps** command to output information about all processes. This action is the default behavior of **ps** on most UNIX platforms. The minimum required access is READ.
SUPERUSER.PROCESS.KILL	Allows user to use the kill() callable service to send signals to any process. The minimum required access is READ.
SUPERUSER.PROCESS.PTRACE	Allows user to use the ptrace() callable service through the dbx debugger to trace any process. The minimum required access is READ. Authorization to the BPX.DEBUG resource is also required to trace processes that run with APF authority or BPX.SERVER authority.
SUPERUSER.SETPRIORITY	Allows user to increase own priority. The minimum required access is READ.
SUPERUSER.SHMMCV.LIMIT	Allows the user to create up to 4,194,304 mutexes or condition variables to be associated with a single shared memory segment. The overall system total of mutexes and condition variables for authorized users must be less than 134,217,729. When authorized applications create the maximum number of mutexes and condition variables, the system requires more auxiliary storage to be available. System dumps that include the OMVS address space also require larger dump data sets to contain the increased size of that address space. It is unlikely that applications will create the maximum number of structures allowed. If the maximum number is created, the increase in auxiliary storage and dump data set size is roughly 350 gigabytes. The minimum required access is READ.

If you are debugging a daemon, use the SUPERUSER.PROCESS.GETPSENT, SUPERUSER.PROCESS.KILL, and SUPERUSER.PROCESS.PTRACE privileges.

Assigning superuser privileges

The example in “Steps for authorizing selected users to transfer ownership of any file” on page 66 applies to the superuser privileges shown in Table 7 on page 63, except the privilege associated with the CHOWN.UNRESTRICTED resource (see “Steps for setting up the CHOWN.UNRESTRICTED profile” on page 67).

Steps for authorizing selected users to transfer ownership of any file

Before you begin, you need to know which users will be assigned superuser authority.

Perform the following steps to authorize selected users to transfer ownership of any file.

1. Define a profile in the UNIXPRIV class to protect the resource called SUPERUSER.FILESYS.CHOWN.

   ```
   RDEFINE UNIXPRIV SUPERUSER.FILESYS.CHOWN UACC(NONE)
   ```

   In general, generic profile names are allowed for resources in the UNIXPRIV class (with a few exceptions, such as SHARED.IDS and FILE.GROUPOWNER.SETGID).

To assign all file system privileges, you can define a profile called SUPERUSER.FILESYS.**.

---

2. Assign selected users or groups as appropriate.

```
PERMIT SUPERUSER.FILESYS.CHOWN CLASS(UNIXPRIV)
       ID(appropriate-groups-and-users) ACCESS(READ)
```

---

3. Activate the UNIXPRIV class, if it is not currently active at your installation.

```
SETROPTS CLASSACT(UNIXPRIV)
```

If you do not activate the UNIXPRIV class and activate SETROPTS RACLIST processing for the UNIXPRIV class, only superusers are allowed to transfer ownership of any file.

---

4. Activate SETROPTS RACLIST processing for the UNIXPRIV class, if it is not already active.

```
SETROPTS RACLIST(UNIXPRIV)
```

If SETROPTS RACLIST processing is already in effect for the UNIXPRIV class, you must refresh SETROPTS RACLIST processing in order for new or changed profiles in the UNIXPRIV class to take effect.

```
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

When you are done, you have authorized selected users to transfer ownership of any file.

Allowing z/OS UNIX users to change file ownerships

On z/OS UNIX systems, superusers can change the ownership of any file to any UID or GID on the system. General users can only change the ownership of files that they own, and only to one of their own associated GIDs. This is considered the more secure implementation, and is the one recommended by IBM. However, you can allow selected z/OS UNIX users to transfer ownership of files they own to any UID or GID on the system.

To allow z/OS UNIX users to transfer ownership of files they own to any UID or GID on the system, create a discrete profile in the UNIXPRIV class called CHOWN.UNRESTRICTED, and permit users with the appropriate access.

Steps for setting up the CHOWN.UNRESTRICTED profile

Before you begin: CHOWN.UNRESTRICTED must be a discrete profile. Matching generic profiles are ignored.

Perform the following steps to set up the CHOWN.UNRESTRICTED profile.

1. Define the discrete profile in the UNIXPRIV class called CHOWN.UNRESTRICTED:

```
RDEFINE UNIXPRIV CHOWN.UNRESTRICTED UACC(NONE)
```

---

2. Permit the user or group with the appropriate access level. UPDATE access is required in order to change ownership to UID 0. READ access is required to change ownership to any other UID value, or to the GID of a group to which the user is not connected. For example:

```
PERMIT CHOWN.UNRESTRICTED CLASS(UNIXPRIV)ID(GRPX) ACCESS(READ)
```

If you do not activate the UNIXPRIV class and activate SETROPTS RACLIST processing for the UNIXPRIV class, only superusers are allowed to transfer ownership of files to others.

---

3. Activate the SETROPTS RACLIST processing for the UNIXPRIV class, if it is not already active.

```
SETROPTS RACLIST(UNIXPRIV)
```

If SETROPTS RACLIST processing is already in effect for the UNIXPRIV class, you must refresh SETROPTS RACLIST processing in order for the CHOWN.UNRESTRICTED profile to take effect.

```
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

When you are done, you have set up the CHOWN.UNRESTRICTED profile in the UNIXPRIV class.

Allowing z/OS UNIX users to search directories

Sometimes z/OS UNIX administrators need the ability to read and search all file system directories to manage file ownerships and permissions. It is not necessary to give such administrators RACF AUDITOR or ROAUDIT authority to provide this ability when directory permission bits and access lists do not explicitly allow access. Instead, you can define a UNIXPRIV profile covering SUPERUSER.FILESYS.DIRSRCH to control such access. This permission is complementary to administrator authorities provided by SUPERUSER.FILESYS.CHOWN and SUPERUSER.FILESYS.CHANGEPERMS.

Note: Use caution when permitting users to the DIRSRCH profile if you employ the strategy of protecting files by disallowing user access to the parent directory. Since users with DIRSRCH profile permission can read and search all directories, their access to files in all subdirectories is determined by the defined file permissions and access lists.

For more information about how to give directory search permission to specified users and groups, see Allowing z/OS UNIX users to read or search directories in *z/OS Security Server RACF Security Administrator's Guide*.

Using the BPX.SUPERUSER resource in the FACILITY class

Using the BPX.SUPERUSER resource in the FACILITY class is another way for users to get the authority to do most of the tasks that require superuser authority.

Steps for setting up BPX.SUPERUSER

Before you begin, you need to know which users need to have superuser authority.

Perform the following steps to set up BPX.SUPERUSER.

1. Define the BPX.SUPERUSER resource in the FACILITY class.

```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
```

You must use the name BPX.SUPERUSER. Substitutions for the name are not allowed.

---

2. If this is the first FACILITY class profile that the installation has defined, activate the FACILITY class with the SETROPTS command.

```
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY)
```

---

3. Permit all users who need superuser authority to this profile. Use the RACF commands shown in the following example, which give the user ID SYSPROG permission to use the su command to obtain superuser authority. It is assumed that the default group for SYSPROG is set up with a GID.

```
ALTUSER SYSPROG OMVS(UID(7) HOME('/u/sysprog') PROGRAM('/bin/sh'))
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(SYSPROG) ACCESS(READ)
```

When you are done, you have set up the BPX.SUPERUSER resource in the FACILITY class and permitted the users who need to have superuser authority. When they need to perform superuser tasks, they can switch to superuser mode using the **su** command or the **Enable superuser mode (SU)** option in the ISPF shell.

Tips:

1. Instead of using BPX.SUPERUSER to permit users to have superuser authority, you could define a group, for example, SUPERUSR. You could then add users who need superuser permission to the group.

   To add the user ID SYSPROG to the SUPERUSR group:

   ```
   CONNECT (SYSPROG) AUTH(USE) GROUP(SUPERUSR) OWNER(SYS1) GRPACC
   ```

   Then permit this group to BPX.SUPERUSER.

   ```
   PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(SUPERUSR) ACCESS(READ)
   ```

2. As an alternative to assigning superuser authority, you can define which superuser attributes a given user is to have, and which system resource limits are to be defined for the user.

Deleting superuser authority

If the installation determines that a user no longer requires superuser authority, the RACF administrator can remove the user from the access list with the PERMIT command.

Example

To remove superuser authority from user ID JOHN:

```
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(JOHN) DELETE
```

Changing a superuser from UID(0) to a unique nonzero UID

Give each user a unique UID and have them use the su command to obtain the authority they need. You can give them the ability to use the su command by giving them READ authority to the BPX.SUPERUSER resource in the FACILITY class. For more information, see su - Change the user ID associated with a session in *z/OS UNIX System Services Command Reference*.

To run SMP/E jobs, the user must have UID(0) or be permitted to the BPX.SUPERUSER resource in the FACILITY class.

Steps for changing a superuser from UID(0) to a unique nonzero UID

Before you begin, you need to know which superusers you want to change from UID(0) to a unique nonzero UID.

Perform the following steps to change a superuser from a UID of 0 to a unique nonzero UID.

1. Change the UID for the superuser from 0 to a unique UID. Base your choice on your particular situation.

Choices for changing superusers	
If you choose this method . . .	**Then . . .**
Have RACF automatically assign an unused UID.	a. Delete the UID from the user's OMVS segment. For example: `ALTUSER JOHN OMVS(NOUID)` b. Issue the ALTUSER command with the AUTOUID keyword. For example: `ALTUSER JOHN OMVS(AUTOUID)` Message IRR52177I identifies the new UID.
Use the ISPF shell to assign the next available UID.	a. Delete the UID from the user's OMVS segment. For example: `ALTUSER JOHN OMVS(NOUID)` b. Assign a new UID, using the ISPF shell. You can display the user's OMVS segment to see the UID that was assigned by the ISHELL. For example: `LISTUSER JOHN OMVS`
Manually assign the UID. If the installation manually manages the UIDs assigned to users, select the next available UID and assign it to the user. For example, to make sure the UID you selected is not already in use by another user, issue: `SEARCH CLASS(USER) UID(7)`	Use the ALTUSER command. For example, assume that the next available UID is 7 and the user ID is JOHN. To reassign the UID, issue: `ALTUSER JOHN OMVS(UID(7))`

_____

2. Permit the user to the BPX.SUPERUSER resource in the FACILITY class.

   For example, for user ID John:

   ```
   PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(JOHN) ACCESS(READ)
   ```

   You can choose to RACLIST the FACILITY class afterward. This step is optional. If you do so, then you will have to do a REFRESH after the user ID is permitted to the FACILITY class. For example:

   ```
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

_____

3. Change the ownership of the user's private files to the new UID. These files are typically those defined in the home directory.

   In the following example, the home directory is `/u/john`. Issue the following command to update the ownership of the files.

   ```
   cd /u/john
   chown -R john /u/john
   ```

The owning UID of the /u/john directory is changed to 7. The owning UID of all files and subdirectories of the /u/john directory is also changed.

Restriction: The chown command requires a UID of 0, the ability to su to 0, or authority to SUPERUSER.FILESYS.CHOWN.

_____

When you are done, you have changed the superuser from a UID of 0 to a unique nonzero UID.

Switching in and out of superuser authority

You can switch in and out of superuser authority. This discussion assumes that the installation has not assigned UID(0) to its superusers. Instead, each user has a unique UID and has been permitted to the BPX.SUPERUSER resource in the FACILITY class.

You can use any of the following methods to gain superuser authority:

- Enter the shell using the OMVS command and then issue the su command with no operands. This creates a nested shell that runs with superuser authority.

  Programs that change the security environment cannot run in a multiprocess address space.

  Tip: When running in this manner, editing a file with the OEDIT command (OEDIT with PF6) returns you to the TSO/E address space where your original authority is still in place.
- Enter the ISPF shell using the ISHELL command or a dialog selection. From the ISPF shell, you can select the option to switch to superuser state. You can then manage the file system using ISPF shell functions while in the superuser state.

  If you enter the ISPF shell, switch to superuser and then exit the ISPF shell, you might lose superuser authority. If the ISPF shell is the only process in the address space, you will lose all connection to kernel services when the ISPF shell terminates. If there is another dubbed process in this address space (for example, another ISPF shell, or a local shell), it will share the UID with the ISPF shell process. For example, you can open an ISPF shell on both sides of a split screen. Therefore, when you toggle to superuser in one ISPF shell, it affects the address space and , both ISPF shells are now superuser. Regardless of which ISPF shell terminates first, the address space retains its UIDs until the ISPF shell is used to toggle back, or the last process is undubbed.
- Enter the shell using rlogin or telnet. Once in the shell, enter the su command to create a nested shell that runs with superuser authority.
- After gaining superuser authority in the ISPF shell, you can split the screen in ISPF and enter the OMVS command. The shell that is started inherits the superuser authority set up in the ISPF shell. For privileged shells (when the effective UID does not match the real UID, or the effective GID does not match the real GID) $HOME/.profile is not run. If the file /etc/suid_profile exists, it will be run.
- If you are permitted to the BPX.SUPERUSER resource, then you can get superuser access through REXX.
- Use the su command from BPXBATCH. Run a job using BPXBATCH following one of these examples that shows a copy of the file:
  - On the PARM= statement, include:

    ```
    SH echo cp /etc/profile /etc/junk | su
    ```

    This pipes the result of the **echo** command (that is, the **copy** command) into the **su** command.
  - With PARM='SH su', code:

    ```
    //STDIN  DD  PATH '/yourpath/input.stuff',PATHOPTS=(ORDONLY)
    ```
  - With no parameters coded at all, create a file that has the su command in it.

    ```
    //BATBPX1  EXEC  PGM=BPXBATCH
    //STDIN  DD PATH='/yourpath/suinput.stuff',PATHOPTS=(ORDONLY)
    ```

In the suinput.stuff section, you would have the **su** command followed by the **copy** command. These are commands as you would have entered them from the console if you had been running in the z/OS UNIX shell.

Also, when you set up your own $HOME/.profile as superuser, specify the /usr/sbin directory in your PATH environment variable because certain superuser utilities are in that directory instead of the /bin directory, such as **automount**. For more information about the profile, see "Customizing $HOME/.profile" on page 200.

Assigning a UID of 0

Although sometimes appropriate, the least desirable method of defining superusers is to assign a UID of 0 in the UID parameter in the OMVS segment of the ADDUSER or ALTUSER commands. In this environment, you risk entering commands that can damage the file system.

If you want to assign a UID of 0, also assign a secondary user ID with a nonzero UID for activities other than system management. For example, you would assign:

```
User ID    SMORG   UID(0)   - used for system maintenance
User ID    SMORG1  UID(7)   - used for regular programming
```

In the following example, the ALTUSER command gives the user ID SMORG superuser authority, makes the root directory the home directory, and causes invocation of the shell in response to a TSO/E OMVS command. If the shell is to be installed, specify the HOME and PROGRAM parameters; if a shell is not to be installed, omit the HOME parameter. This user must be in a RACF group, typically SYS1, and the group must have an OMVS GID associated with it.

```
ALTUSER   SMORG  OMVS(UID(0)  HOME('/')   PROGRAM('/bin/sh'))
ALTGROUP  SYS1   OMVS(GID(0))
```

You might choose to assign UID(0) to multiple RACF user IDs. However, try to minimize the use of UID(0). Assignment of UID(0) should be limited to the user associated with started procedures such as the OMVS kernel and the user who installs z/OS UNIX. It should be avoided for the user IDs belonging to the real users whenever possible.

If the SHARED.IDS profile is defined in the UNIXPRIV class, you might need to use the SHARED keyword because UID(0) is likely to be used by several IDs. For example:

```
ALTUSER SMORG OMVS(UID(0) SHARED HOME('/') PROGRAM('/bin/sh'))
```

Setting up the UNIX-related FACILITY and SURROGAT class profiles

You can control who can use certain UNIX functions when you define RACF profiles with UACC(NONE) to protect the appropriate resources in the FACILITY and SURROGAT classes. The resources that are related to UNIX functions start with the prefix BPX. Generally, authorized users need at least READ access to the FACILITY resources in order to use the UNIX function.

Do not define the generic profile BPX.* or unintended security-related behavior might occur. If BPX.* is defined, then the OMVS address space identity must be permitted to it and BPXOINIT must have a different user identity than OMVS. Following these guidelines will prevent unintended security-related behavior from occurring on your system.

To activate RACF control of UNIX functions, use the RACF SETROPTS CLASSACT FACILITY command. Permit your authorized users to the appropriate resources before you activate the FACILITY class or else users will not be able to use protected UNIX functions.

Because TRUSTED users are not permitted to the BPX.SERVER or the BPX.DAEMON profiles by default, they do not have any authorities that are associated with having access to these two profiles.

Defining class profiles for security reasons

For security reasons, you might need to define these class profiles. All of the following are FACILITY class profiles, except for BPX.SRV, which is a SURROGAT class profile.

Table 8. Defining class profiles for security reasons

FACILITY class profile	Description
BPX.CF	Controls access to the _cpl service.
BPX.CONSOLE	Allows a permitted user the ability to use the _console() or _console2() services.
BPX.DAEMON	BPX.DAEMON serves two functions in the z/OS UNIX environment: • Any superuser that is permitted to this profile has the daemon authority to change MVS identities via z/OS UNIX services without knowing the target user ID's password or password phrase. This identity change can only occur if the target user ID has a defined OMVS segment. If BPX.DAEMON is not defined, then all superusers (UID=0) have daemon authority. If you want to limit which superusers have daemon authority, define this profile and permit only selected superusers to it. • Any program that is loaded into an address space that requires daemon level authority must be defined to program control. If the BPX.DAEMON FACILITY class profile is defined, then z/OS UNIX will verify that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed: – seteuid – setuid – setreuid – pthread_security_np() – auth_check_resource_np() – _login() – _spawn() with user ID change – _passwd() Daemon authority is required only when a program does a setuid(), seteuid(), setreuid(), or spawn() user ID to change the current UID without first having issued a _passwd() call to the target user ID. In order to change the MVS identity without knowing the target user ID's password or password phrase, the caller of these services must be a superuser. Additionally, if a BPX.DAEMON FACILITY class profile is defined and the FACILITY class is active, the caller must be permitted to use this profile. If a program comes from a controlled library and knows the target UID's password or password phrase, it can change the UID without having daemon authority. The RACF WARNING mode is not supported for BPX.DAEMON. For more information about BPX.DAEMON, see "Establishing the correct level of security for daemons" on page 313.
BPX.DAEMON.HFSCTL	Controls which users with daemon authority are allowed to load uncontrolled programs from MVS libraries into their address space. **Restriction:** BPX.DAEMON.HFSCTL does not allow generic profiles.
BPX.DEBUG	Users with READ access to BPX.DEBUG can debug certain types of restricted processes. These do not include processes that have a PID of 1. To debug programs that run with APF authority or with BPX.SERVER authority, they can use **dbx** to call the ptrace callable service.

Table 8. Defining class profiles for security reasons (continued)

FACILITY class profile	Description
BPX.EXECMVSAPF.*program_name*	Allows unauthorized callers of the execmvs callable service to pass an argument that is greater than 100 characters to an authorized program. If the FACILITY class resource exists, then unauthorized callers can pass arguments greater than 100 characters to the program name that is specified in the FACILITY class profile. Individual users do not need to be given access to the profile. If you do not want unauthorized callers to pass an argument greater than 100 characters to any authorized programs, do not define any BPX.EXECMVSAPF.*program_name* profiles. To allow certain authorized programs to be called with an argument greater than 100 characters, define a profile for each program: `BPX.EXECMVSAPF.YOURPGM` `BPX.EXECMVSAPF.MYPGM` To allow a group of commonly named authorized programs to be called with an argument greater than 100 characters, define a profile that allows for pattern matching. For example, if you have a set of related programs that all begin with the same three characters, MYP, define: `BPX.EXECMVSAPF.MYP*` As a result, all unauthorized callers can pass an argument greater than 100 characters to any authorized program that begins with the characters MYP. To allow all unauthorized users the ability to pass any argument up to 4096 characters long to any authorized program, then define one profile: `BPX.EXECMVSAPF.*` However, IBM does not recommend defining this type of profile.
BPX.FILEATTR.APF	Controls which users are allowed to set the APF-authorized attribute in a z/OS UNIX file. This authority allows the user to create a program that will run APF-authorized. This is similar to the authority of allowing a programmer to update SYS1.LINKLIB or SYS1.LPALIB.
BPX.FILEATTR.PROGCTL	Controls which users are allowed to set the program control attribute. Programs marked with this attribute can execute in server address spaces that run with a high level of authority. See “Defining programs in UNIX files to program control” on page 317 for more information.
BPX.FILEATTR.SHARELIB	Indicates that extra privilege is required when setting the shared library extended attribute via the chattr() callable service. This setting prevents the shared library region from being misused. See “Defining UNIX files as shared library programs” on page 319 for more information.
BPX.JOBNAME	Controls which users are allowed to set their own job names by using the _BPX_JOBNAME environment variable or the inheritance structure on spawn. Users with READ or higher permissions to this profile can define their own job names.
BPX.MAINCHECK	Extends the enhanced program security protection to your UNIX daemons and servers that do not use RACF execute-controlled programs. For more information, see “RACF with enhanced program security, BPX.DAEMON, and BPX.MAINCHECK” on page 313 and “RACF with enhanced program security, BPX.SERVER, and BPX.MAINCHECK” on page 344. **Restriction:** BPX.MAINCHECK does not allow generic profiles.
BPX.MAP	Controls access to the _map and _map_init services.

Table 8. Defining class profiles for security reasons (continued)	
FACILITY class profile	**Description**
BPX.NEXT.USER	Enables automatic assignment of UIDs and GIDs. The APPLDATA field of this profile specifies a starting value, or range of values, from which RACF will derive unused UID and GID values. For more information about BPX.NEXT.USER, see BPX.NEXT.USER in *z/OS Security Server RACF Security Administrator's Guide*.
BPX.POE	Controls access to the _poe service.
BPX.SAFFASTPATH	Enables faster security checks for file system and IPC constructs. For more information, see “Fastpath support for System Authorization Facility (SAF) ” on page 288. **Restriction:** BPX.SAFFASTPATH does not allow generic profiles.
BPX.SERVER	Restricts the use of the pthread_security_np() service. A user with at least READ or WRITE access to the BPX.SERVER FACILITY class profile can use this service. It creates or deletes the security environment for the caller's thread. This profile is also used to restrict the use of the BPX1ACK service, which determines access authority to z/OS resources Servers with authority to BPX.SERVER must run in a clean program-controlled environment. z/OS UNIX will verify that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed: • seteuid • setuid • setreuid • pthread_security_np() • auth_check_resource_np() • _login() • _spawn() with user ID change • _passwd() For more information about BPX.SERVER, see Chapter 18, “Preparing security for servers,” on page 341 and “Establishing the correct level of security for daemons” on page 313.

Table 8. Defining class profiles for security reasons (continued)

FACILITY class profile	Description
BPX.SMF or BPX.SMF.*type*.*subtype*	Grants a permitted user access to write an SMF record or to test if an SMF type or subtype is being recorded. • The BPX.SMF profile grants the permitted user the authority to write or test for any SMF record that is being recorded. • For more granular access to writing SMF records, BPX.SMF.*type*.*subtype* grants a permitted user the authority to write or test only the SMF record of the specific type and subtype contained in the FACILITY class profile name. The program-controlled attribute is not required when using the BPX.SMF resource profile. If SMFUPDATE(ON) was specified in the active BPXPRMxx member, processes permitted to the BPX.SMF.*type*.*subtype* resource profile with UPDATE access authority are also exempt from the program-controlled attribute requirement. Users permitted to the BPX.SMF.*type*.*subtype* FACILITY class profile with READ access authority require a clean program-controlled environment. The smf_record syscall verifies that the address space has not loaded any executables that are uncontrolled and any future loads or execs to files that reside in uncontrolled libraries are prevented. Note that type and subtype in the FACILITY class name do not have leading zeros. Some examples are as follows: – BPX.SMF.7.0 – BPX.SMF.119.94 – BPX.SMF.0.0
BPX.SHUTDOWN	Controls access to the oe_env_np service to register and block for OMVS shutdown.
BPX.SRV.*userid*	Allows users to change their UID if they have access to BPX.SRV.*userid*, where *userid* is the MVS user ID associated with the target UID. BPX.SRV.*userid* is a RACF SURROGAT class profile.
BPX.STOR.SWAP	Controls which users can make address spaces nonswappable. Users who are permitted with at least READ access to BPX.STOR.SWAP can invoke the __mlockall() callable service to make their address space either nonswappable or swappable. When an application makes an address space nonswappable, it might cause additional real storage in the system to be converted to preferred storage. Because preferred storage cannot be configured offline, using this service can reduce the installation's ability to reconfigure storage in the future. Any application that uses this service should warn the customer about this side effect in their installation documentation.
BPX.STICKYSUG.*program_name*	Enables the exec and spawn services to use the MVS program search order to locate the program to be run when the specified path name resolves to a file with the sticky attribute and either the set-user-id or set-group-id attributes. If a FACILITY class resource exists, then the MVS program search order can be used in locating the program name that is specified in the FACILITY class profile. Individual users do not need to be given access to the profile. For examples of using this class profile, see “Examples of BPX.STICKYSUG.program_name” on page 77.
BPX.SUPERUSER	Allows users to switch to superuser authority. For more information about BPX.SUPERUSER, see “Superusers in z/OS UNIX” on page 62.
BPX.UNLIMITED.OUTPUT	Allows users to use the _BPX_UNLIMITED_OUTPUT environment variable to override the default spooled output limits for processes.

Table 8. Defining class profiles for security reasons (continued)	
FACILITY class profile	**Description**
BPX.WLMSERVER	Controls access to the WLM server functions _server_init() and _server_pwu(). It also controls access to these C language WLM interfaces: • QuerySchEnv() • CheckSchEnv() • DisconnectServer() • DeleteWorkUnit() • JoinWorkUnit() • LeaveWorkUnit() • ConnectWorkMgr() • CreateWorkUnit() • ContinueWorkUnit() A server application with read permission to this FACILITY class profile can use both the server functions and the WLM C language functions to create and manage work requests.

Examples of BPX.STICKYSUG.*program_name*

1. If you do not want the exec and spawn services to use the MVS program search order to locate programs, do not define any BPX.STICKYSUG.*program_name* profiles.
2. If you want the exec and spawn services to use the MVS program search order for certain programs, then define a profile for each program:

   ```
   BPX.STICKYSUG.YOURPGM
   BPX.STICKYSUG.MYPGM
   ```

3. If you want the exec and spawn services to use the MVS program search order for a group of commonly named programs, then define a generic profile:

   ```
   BPX.STICKYSUG.MYP*
   ```

   The exec and spawn will use the MVS program search order for any programs that begin with the characters MYP.
4. If you want the exec and spawn services to always use the MVS program search order, then define one profile:

   ```
   BPX.STICKYSUG.*
   ```

   However, IBM does not recommend defining this type of profile.

Permissions for undefined FACILITY class profiles

Table 9 on page 77 shows whether the caller is permitted to use the services with the indicated profile if that profile is defined and if the caller's user ID is permitted to the specified RACF FACILITY class profile.

- YES indicates that the caller is permitted to use the services that are associated with the profile.
- NO indicates that the caller is not permitted to use the services that are associated with the profile.

For example, if BPX.DAEMON is not defined and the caller has a nonzero UID, then that caller would not be permitted to use setuid.

Table 9. Permissions for undefined FACILITY class profiles		
Undefined FACILITY class profile	**If UID(0)**	**If not UID(0)**
BPX.CF	No	No

Table 9. Permissions for undefined FACILITY class profiles (continued)		
Undefined FACILITY class profile	**If UID(0)**	**If not UID(0)**
BPX.CONSOLE. It controls access to authorized features of the _console() service and not used to control which users can use the base _console() service.	Yes	No
BPX.DAEMON	Yes	No
BPX.DAEMON.HFSCTL	No	No
BPX.DEBUG	No	No
BPX.EXECMVSAPF.*program_name*	No	No
BPX.FILEATTR.APF	No	No
BPX.FILEATTR.PROGCTL	No	No
BPX.FILEATTR.SHARELIB	No	No
BPX.JOBNAME	Yes	No
BPX.MAINCHECK	No	No
BPX.MAP	Yes	No
BPX.NEXT.USER, which is used by RACF to assign UIDs and GIDs when creating or altering a user ID's OMVS segment and is not processed directly by z/OS UNIX.	Not applicable	Not applicable
BPX.UNLIMITED.OUTPUT	Yes	No
BPX.POE	Yes	No
BPX.SAFFASTPATH	No	No
BPX.SERVER	Yes	No
BPX.SHUTDOWN	Yes	No
BPX.SMF or BPX.SMF.*type.subtype*	No	No
BPX.SRV.*userid*. Its profiles are defined in the RACF SURROGAT class.	No	No
BPX.STOR.SWAP	Yes	No
BPX.STICKYSUG.*program_name*	No	No
BPX.SUPERUSER	No	No
BPX.WLMSERVER	Yes	No

Permissions for defined FACILITY class profiles if user ID is not permitted

Table 10 on page 78 shows whether the caller is permitted to use the services with the indicated profile if that profile is defined and the caller's user ID is not permitted to the specified RACF FACILITY class profile.

- YES indicates that the caller is permitted to use the services that are associated with the profile.
- NO indicates that the caller is not permitted to use the services that are associated with the profile.

Table 10. Permissions for defined FACILITY class profiles if user ID is not permitted		
Defined FACILITY class profile and caller is not permitted	**If UID(0)**	**If not UID(0)**
BPX.CF	No	No
BPX.CONSOLE. It controls access to authorized features of the _console() service and not used to control which users can use the base _console() service.	Yes	No
BPX.DAEMON	No	No
BPX.DAEMON.HFSCTL	No	No
BPX.DEBUG	No	No
BPX.EXECMVSAPF.*program_name*	Yes	Yes
BPX.FILEATTR.APF	No	No
BPX.FILEATTR.PROGCTL	No	No
BPX.FILEATTR.SHARELIB	No	No

Table 10. Permissions for defined FACILITY class profiles if user ID is not permitted (continued)		
Defined FACILITY class profile and caller is not permitted	**If UID(0)**	**If not UID(0)**
BPX.JOBNAME	Yes	No
BPX.MAINCHECK	Yes	Yes
BPX.MAP	No	No
BPX.NEXT.USER, which is used by RACF to assign UIDs and GIDs when creating or altering a user ID's OMVS segment and is not processed directly by z/OS UNIX.	Not applicable	Not applicable
BPX.UNLIMITED.OUTPUT	Yes	No
BPX.POE	No	No
BPX.SAFFASTPATH	No	No
BPX.SERVER	No	No
BPX.SHUTDOWN	No	No
BPX.SMF or BPX.SMF.*type.subtype*	No	No
BPX.SRV.*userid*. Its profiles are defined in the RACF SURROGAT class.	No	No
BPX.STOR.SWAP	No	No
BPX.STICKYSUG.*program_name*	Yes	Yes
BPX.SUPERUSER	No	No
BPX.WLMSERVER	No	No

Permission for defined FACILITY class profiles if user ID is permitted

Table 11 on page 79 shows whether the caller is permitted to use the services with the indicated profile if that profile is defined and the caller's user ID is permitted to the specified RACF FACILITY class profile.

- YES indicates that the caller is permitted to use the services associated with the profile.
- NO indicates that the caller is not permitted to use the services that are associated with the profile.

Table 11. Permissions for defined FACILITY class profiles if user ID is permitted		
Defined FACILITY class profile and caller is permitted	**If UID(0)**	**If not UID(0)**
BPX.CF	Yes	Yes
BPX.CONSOLE. It controls access to authorized features of the _console() service and not used to control which users can use the base _console() service.	Yes	Yes
BPX.DAEMON	Yes	No
BPX.DAEMON.HFSCTL	Yes	Yes
BPX.DEBUG	Yes	Yes
BPX.EXECMVSAPF.*program_name*	Yes	Yes
BPX.FILEATTR.APF	Yes	Yes
BPX.FILEATTR.PROGCTL	Yes	Yes
BPX.FILEATTR.SHARELIB	Yes	Yes
BPX.JOBNAME	Yes	Yes
BPX.MAINCHECK	Yes	Yes
BPX.MAP	Yes	Yes
BPX.NEXT.USER, which is used by RACF to assign UIDs and GIDs when creating or altering a user ID's OMVS segment and is not processed directly by z/OS UNIX.	Not applicable	Not applicable
BPX.UNLIMITED.OUTPUT	Yes	Yes
BPX.POE	Yes	Yes
BPX.SAFFASTPATH	Yes	Yes
BPX.SERVER	Yes	Yes
BPX.SHUTDOWN	Yes	Yes

Table 11. Permissions for defined FACILITY class profiles if user ID is permitted (continued)		
Defined FACILITY class profile and caller is permitted	**If UID(0)**	**If not UID(0)**
BPX.SMF or BPX.SMF.*type.subtype*	Yes	Yes
BPX.SRV.*userid*. Its profiles are defined in the RACF SURROGAT class.	Yes	Yes
BPX.STOR.SWAP	Yes	Yes
BPX.STICKYSUG.*program_name*	Yes	Yes
BPX.SUPERUSER	Yes	Yes
BPX.WLMSERVER	Yes	Yes

Security requirements for ServerPac and CBPDO installation

Before you can do the ServerPac or CBPDO installation, or install maintenance, you must satisfy certain security requirements.

1. The user ID must be UID=0 or permitted to the BPX.SUPERUSER resource in the FACILITY class, and be connected to a group that has a GID.
2. The user ID must be permitted READ access to the BPX.FILEATTR.SHARELIB, BPX.FILEATTR.APF, and BPX.FILEATTR.PROGCTL resources in the FACILITY class (or BPX.FILEATTR.* if you choose to use a generic name for both resources). These commands are also provided in SYS1.SAMPLIB(BPXISEC1).

   To define BPX.FILEATTR.SHARELIB, BPX.FILEATTR.APF, and BPX.FILEATTR.PROGCTL, issue:

   ```
   RDEFINE FACILITY BPX.FILEATTR.SHARELIB UACC(NONE)
   RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
   RDEFINE FACILITY BPX.FILEATTR.PROGCTL UACC(NONE)
   SETROPTS CLASSACT(FACILITY)
   SETROPTS RACLIST(FACILITY)
   ```

   These commands are also provided in SYS1.SAMPLIB.

   ```
   PERMIT BPX.FILEATTR.SHARELIB CLASS(FACILITY) ID(your_userid) ACCESS(READ)
   PERMIT BPX.FILEATTR.APF CLASS(FACILITY) ID(your_userid) ACCESS(READ)
   PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(your_userid) ACCESS(READ)
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

   Or, if you choose to use a generic facility:

   ```
   SETROPTS GENERIC(FACILITY)
   RDEFINE FACILITY BPX.FILEATTR.* UACC(NONE)
   SETROPTS CLASSACT(FACILITY)
   SETROPTS RACLIST(FACILITY)
   ```

   ```
   PERMIT BPX.FILEATTR.* CLASS(FACILITY) ID(your_userid) ACCESS(READ)
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

3. Define the following user ID and group IDs in your security data base. Even though they are lowercase in the example, these names should be defined in uppercase for ease of use and manageability.

   For group IDs:

   - *uucpg*
   - *TTY*

   For user IDs:

   - *uucp*

   Rules:

   a. The GID and UID values assigned to these IDs cannot be used by any other IDs. They must be unique. If you assign the same GID to multiple groups, control at an individual group level is lost because the GID is used in z/OS UNIX security checks. Because RACF groups that have the same GID assignment are treated as a single group during the z/OS UNIX security checks, the sharing of

resources between groups might happen unintentionally. Likewise, the sharing of UIDs allows each user access to all of the resources associated with the other users of that shared UID. The shared access includes not only z/OS UNIX resources such as files, but also includes the possibility that one user could access UNIX resources of the other user that are normally considered to be outside the scope of z/OS UNIX.

b. You must duplicate the required user ID and group names in each security database, including the same UID and GID values in the OMVS segment. Duplicating the IDs simplifies the process of transporting the zFS data sets from test systems to production systems. For example, the group name *TTY* on System 1 must have the same GID value on System 2 and System 3. If you cannot synchronize your databases, you will need to continue running the FOMISCHO job against each system after z/OS UNIX is installed.

The following topics describe how to define these IDs to RACF. (If you are using an equivalent security product, refer to that product's documentation.) All the RACF commands are issued by a user ID with RACF SPECIAL authority. Three procedures are described:

- "If you use uppercase group and user IDs" on page 81
- "If you use mixed-case group and user IDs" on page 81
- "If you have problems with names such as UUCP, UUCPG, and TTY" on page 82

If you use uppercase group and user IDs

If you use only uppercase group and user IDs on your system, RACF users can use the BPX1SEC1 sample in SAMPLIB. They can also use the ADDGROUP or ADDUSER commands to define the group IDs and user IDs, as shown in the following examples.

1. To define the TTY group:

   ```
   ADDGROUP TTY (OMVS(GID(2))
   ```

   where 2 is an example of a unique group ID on your system. Do not connect users to this group. This is the same group that is specified on the TTYGROUP statement in the BPXPRMxx member on your target system.

2. To define the UUCPG group:

   ```
   ADDGROUP UUCPG OMVS(GID(8765))
   ```

   where 8765 is an example of a unique group ID on your system.

3. To define the UUCP user ID, issue:

   ```
   ADDUSER UUCP DFLTGRP(UUCPG) PASSWORD(xxxxxxx)
   OMVS(UID(396) HOME('/usr/spool/uucppublic')
   PROGRAM('/bin/sh'))
   ```

   where

   - 396 is an example of a unique OMVS UID. Do not use UID(0).
   - `HOME('/usr/spool/uucppublic')` is a required parameter that specifies the initial directory path name for the user ID.
   - `PROGRAM('/bin/sh')` is a required parameter that specifies the path name in the shell program for the user ID.

If you use mixed-case group and user IDs

If you already use mixed-case group and user IDs on your system and the user (*uucp*) and group (*uupcg*) do not conflict with existing names, perform the steps for uppercase IDs in "If you use uppercase group and user IDs" on page 81.

It is not necessary to add the lowercase or mixed-case names to your alias table, mapping them to uppercase. Using the alias table degrades performance and increases systems management and complexity. When lowercase or mixed-case names are not found in the alias table, or there is no table active, they are folded to uppercase. For more information about the alias table, see "USERIDALIASTABLE" on page 33.

If you have problems with names such as UUCP, UUCPG, and TTY

If names such as uucp, uucpg, and TTY are not allowed on your system (or if they conflict with existing names), the following examples show the RACF commands to define the group ID and user IDs.

1. To define a group ID instead of the TTY group, issue:

   ```
   ADDGROUP xxtty OMVS(GID(2))
   ```

   where 2 is an example of a unique group ID on your system, and XXTTY is replaced by a 1-to 8-character group ID of your choice. Do not connect users to this group. This would be the same group name to be specified in the TTYGROUP statement in the BPXPRMxx member on your target system.
2. To define a group ID instead of the UUCPG group, issue:

   ```
   ADDGROUP xxuucpg OMVS(GID(8765))
   ```

   where 8765 is an example of a unique group ID on your system, and *xxuucpg* is replaced by a 1-to 8-character group ID of your choice.
3. To define a user ID instead of the UUCP user ID, issue:

   ```
   ADDUSER xxuucp DFLTGRP(UUCPG) PASSWORD(xxxxxxx)
   OMVS(UID(396) HOME('/usr/spool/uucppublic')
   PROGRAM('/bin/sh'))
   ```

   where:

   - 396 is an example of a unique UID. Do not use UID(0).
   - xxuucp is replaced by a user ID of your choice. This is a normal user ID which owns all the UUCP files and directories. Use this user ID when editing configuration files or performing other administrative tasks.
   - HOME('/usr/spool/uucppublic') is a required parameter that specifies the initial path name for the user ID.
   - PROGRAM('/bin/sh') is a required parameter that specifies the path name in the shell program for the user ID.
4. Set up a user ID alias table.

   Using the alias table reduces performance and increases systems management costs and complexity. For more information about the alias table, see "USERIDALIASTABLE" on page 33.

   If you do not have a user ID alias table defined, you will need to create one. Create it first on your driving system and then on any system image using this product. This fits in with the IBM strategy to place all customized data in the /etc directory. This table is specified by the USERIDALIASTABLE keyword in the BPXPRMxx member. Because the user ID name alias table must be protected from update by nonprivileged users, only users with superuser authority should be given update access to it. All users should have read access to the file.

   Your user ID alias table will need to contain your MVS chosen names and the associated required names. Your chosen MVS user ID and group names must be located in columns 1-8 and the associated aliases must be located on the same line in columns 10-17.

   ```
   :groups
   XXTTY        TTY
   XXUUCPG  uucpg
   :userids
   XXUUCP     uucp
   ```

5. Activate the user ID alias table. If you are already using the user ID alias table, new database queries will yield the new alias if the userid performing the query has read/execute access to the userid/group name alias table. The table is checked every 15 minutes and refreshed if it has been changed. If a change needs to be activated sooner, you can use the SETOMVS or SET OMVS operator commands.

   If you are not using the user ID alias table, you can use the SET OMVS operator command to activate it now. For example:

   ```
   SET OMVS USERIDALIASTABLE=/etc/tablename
   ```

   where `/etc/tablename` is the name of your user ID alias table. You can also use the SETOMVS operator command.
6. Specify USERIDALIASTABLE in your BPXPRMxx member to make this change permanent for your next IPL.
7. Perform these tasks on all of your driving, test, and production system images.

For more information, see:

- "Defining z/OS UNIX users to RACF" on page 51
- "Defining group identifiers (GIDs)" on page 57

Defining cataloged procedures to RACF

If a cataloged procedure starts a program that uses z/OS UNIX or its resources, the procedure should be defined to RACF. An example is the Resource Measurement Facility (RMF) Monitor III Gatherer (RMFGAT).

The RMFGAT started task must be associated with a user ID using ICHRINO3 or the STARTED class, and the user ID that you assign to it must be defined to RACF and needs to have a UID. The user ID must also belong to a group that has a GID. You can use the user ID RMFGAT, but it can be any RACF-defined user ID.

Example

The following example gives RMFGAT a UID of 123 and designates the root directory as its home directory:

```
ADDUSER RMFGAT DFLTGRP(OMVSGRP) OMVS(UID(123) HOME('/')) NOPASSWORD
```

Controlling access to files and directories

The system provides security for local files by verifying that a z/OS UNIX user can access a directory, a file, and every directory in the path to the file.

The system does a security check for a file, FIFO special file (named pipe), character special file, and directory. It does not check an unnamed pipe, because this pipe can be accessed only by the parent process that created the pipe and by child processes of the creating process. When the last process using an unnamed pipe closes it, the pipe vanishes.

Every file and directory has security information, which consists of:

- File access permissions (including an ACL, if one exists)
- UID and GID of the file
- Audit options that the file owner can control
- Audit options that the security auditor can control

The file access permission bits that accompany each file provide discretionary access control (DAC). These bits determine the type of access a user has to a file or directory.

The following topics assume that ACLs are not being used. Go to "Using access control lists (ACLs)" on page 89 for more information about ACLs.

Setting classes for a user's process

The access permission bits are set for three classes. When a user's process accesses a file, the system determines the class of the process and then uses the permission bits for that class to determine if the process can access the file. For a file, a process can be in only one class. The class for a process can be different for each file or directory.

The class is one of the following:

Owner class
Any process with an effective UID that matches the UID of the file.

Group class
Any process with an effective GID or supplemental group GID that matches the GID of the file when the UIDs do not match.

Other class
Any process that is not in the owner or group class, such as when the UIDs or GIDs do not match.

By default, the system sets the UID and GID of the file when the file is created:

- The UID is set to the effective UID of the creating process.
- The GID is set to the GID of the owning directory. You can define FILE.GROUPOWNER.SETGID to change this behavior; see “Steps for setting up the FILE.GROUPOWNER.SETGID profile” on page 84.

To change the UID of a file, a person with superuser authority, or the file owner with appropriate access to the CHOWN.UNRESTRICTED profile in the UNIXPRIV class, can enter a **chown** command or use the chown() callable service. To change the GID of a file, a superuser or the file owner (that is, a process in the owner class) can enter a **chgrp** command or use the chgrp() function. You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges, as explained in “Using UNIXPRIV class profiles ” on page 63.

If you want to specify that, by default, the group owner of a new file is to come from the effective GID of the creating process, you need to set up a profile in the UNIXPRIV class called FILE.GROUPOWNER.SETGID. “Steps for setting up the FILE.GROUPOWNER.SETGID profile” on page 84 describes the process.

Steps for setting up the FILE.GROUPOWNER.SETGID profile

Perform the following steps to set up the FILE.GROUPOWNER.SETGID profile.

1. Define the FILE.GROUPOWNER.SETGID profile.

   ```
   RDEFINE UNIXPRIV FILE.GROUPOWNER.SETGID
   ```

   ——

2. Activate the UNIXPRIV class, if it is not currently active at your installation.

   ```
   SETROPTS CLASSACT(UNIXPRIV)
   ```

   ——

3. Activate the SETROPTS RACLIST processing for the UNIXPRIV class, if it is not already active.

   ```
   SETROPTS RACLIST(UNIXPRIV)
   ```

   If SETROPTS RACLIST processing is already in effect for the UNIXPRIV class, you must refresh SETROPTS RACLIST processing in order for the FILE.GROUPOWNER.SETGID profile to take effect.

   ```
    SETROPTS RACLIST(UNIXPRIV) REFRESH
   ```

   ——

When you are done, you have set up the FILE.GROUPOWNER.SETGID profile. The set-gid bit for a directory determines how the group owner is initialized for new objects created within the directory.

- If the set-gid bit is on, then the owning GID is set to that of the directory.
- If the set-gid bit is off, then the owning GID is set to the effective GID of the process.

Important: When a new file system is mounted, you must turn on the set-gid bit of its root directory if you want new objects within the file system to have their group owner set to that of the parent directory.

Accessing files

To access local files, users need the following permissions:

- Read and search permission to all directories in the path names of files the user should use. Read permission is required for some options of some commands.
- Write permission to all directories in which the user will be creating or deleting files or directories.
- Read permission, write permission, or read and write permission, as appropriate to all files that the user needs to access.
- Execute permission to executable files that the user needs to run.

Table 12 on page 85 shows types of access and the permissions granted by the accesses.

Table 12. File access types and permission bits. This table shows the permissions needed for each access type.

Access	Permission for file	Permission for directory
Read	Permission to read or print the contents.	Permission to read, but not search, the contents.
Write	Permission to change, add to, or delete from the contents.	Permission to change, add, or delete directory entries.
Execute or Search	Permission to run the file. This permission is used for executable files.	Permission to search the directory.

With read permission, you can see the names of the entries stored in the directory but you cannot see the attributes stored in the entries nor access the contents of the directory. With search permission, you can read the attributes from a specific entry and locate a specific entry of the directory.

Changing the permission bits for a file

To change the permission bits for a file, use one of the following:

- The ISPF shell
- The `chmod` command. You can use it to change individual bits without affecting the other bits. You can also use the `setfacl` command to change permission bits (see "Managing ACLs" on page 89).
- The chmod() callable service in a program. The callable service changes all permission bits to the values in the *mode* argument.

The file owner or a superuser can use the `chmod` command or chmod() callable service, or you can define a profile in the UNIXPRIV class to grant RACF authorization. The file mode creation mask does not affect the permission value that was specified by either `chmod` or chmod().

Changing the owner or group for a file

An interactive user might need to change the UID or GID for a file. To protect the data in a file from unauthorized users, the system controls who can change the file access:

- To change the owner and, optionally, the group, the superuser can enter a `chown` command. The new owner can be identified with a user ID or a UID. The group, if specified, can be identified with a RACF group name or a GID.

The CHOWN.UNRESTRICTED profile allows users to use the chown command to transfer ownership of their own files. SUPERUSER.FILESYS.CHOWN allows users to use chown to change ownership of any file.

- To change the group owner to a specified GID, the superuser or the file owner can enter a `chgrp` command. The new group can be identified with a group ID.

Creating a set-user-ID or set-group-ID executable file

A superuser or the file owner can use a chmod command or chmod() callable service to change two options for an executable file. The options are set in two file mode bits:

- Set-user-ID (S_ISUID) with the setuid option
- Set-group-ID (S_ISGID) with the setgid option

If one or both of these bits are on, the effective UID, effective GID, or both, plus the saved UID, saved GID, or both, for the process running the program are changed to the owning UID, GID, or both, for the file. This change temporarily gives the process running the program access to data the file owner or group can access.

In a new file, both bits are set `off`. Also, if the owning UID or GID of a file is changed or if the file is written in, the bits are turned off.

In shell scripts, these bits are ignored.

Protecting data

Local files and directories are protected by RACF security rules. You can use permission bits to control access; access control lists (ACLs) can also be used in conjunction with permission bits. For more information, see “Using access control lists (ACLs)” on page 89.

Permission bit information is stored in the file security packet (FSP) within each file and directory. (ACLs can also be stored with the file.) Permission bits allow you to specify read authority, write authority, or search authority for a directory. They also allow specification of read, write, or execute authority for a file. Because there are three sets of bits, separate authorities can be specified for the owner of the file or directory, the owning group, and everyone else (such as RACF's universal access authority, or UACC). The owner is represented by a UID. The owning group is represented by a GID. Access checking compares the user's UID and GID to the ones stored in the FSP.

When a security decision is needed, the file system calls RACF and supplies the FSP (and ACL, if one exists). RACF makes the decision, does any auditing, and returns control to the file system. RACF does not provide commands to maintain the FSP (and ACL). System Authorization Facility (SAF) services handle the FSP (and ACL) maintenance. z/OS UNIX provides commands that invoke these SAF services.

For information about using RACF authorization to grant privileges for use of local files and directories, see Table 7 on page 63.

Obtaining security information for a file

Users with search access to the directories in the path name and, for some options, read access to the directories can check a file's security information, including the access permissions. They do not need read access to the file being checked. Programs can also check security information for files.

To check the security information, do one of the following:

- Use the ISPF shell
- Enter the `ls -l` or `ls -E` shell command.
- Run a stat() or fstat() callable service in a program.

In response, the system displays the user ID and the RACF group name that correspond to the file's UID and GID. The system displays the UID and GID only if it cannot find the corresponding user ID and RACF group name.

For `ls -l`, the permission bits appear as 11 characters.

```
tfffgggoooa
```

Table 13 on page 87 explains the meanings of each character.

Table 13. Explanation of the characters in tfffgggoooa format. This table lists the meanings of each character.

Character	Meaning
t	Identifies the type of file or directory: **—** Regular file **b** Block special file (not supported for z/OS UNIX) **c** Character special file **d** Directory **e** External link **l** Symbolic link **p** FIFO special file **s** Socket file type
fff	Owner permissions • First character: Read access • Second character: Write access • Third character: Execute or, for a directory, search
ggg	Group permissions • First character: Read access • Second character: Write access • Third character: Execute or, for a directory, search
ooo	Other permissions • First character: Read access • Second character: Write access • Third character: Execute or, for a directory, search
a	If a is a plus sign, then the file contains extended ACL entries. Use the `getfacl` command to display the ACL entries.

The permissions `fff`, ggg, and ooo are displayed as shown in Table 14 on page 88.

Table 14. Explanation of the characters in fff, ggg, and ooo format. This table lists the meanings and position for each character for the fff permission.

Character	Position	Meaning
–	Any	No access
r	First	Read access
w	Second	Write access
x	Third	Execute (or, for a directory, search)
s	Third (owner only)	Execute permission for owner, set-user-ID set
S	Third (owner only)	No execute permission for owner, set-user-ID set
s	Third (group only)	Execute permission for group, set-group-ID set
S	Third (group only)	No execute permission for group, set-group-ID set
t	Third (other only)	Execute permission for other, sticky bit set
T	Third (other only)	No execute permission for other, with sticky bit set

For example, `rwx` means read, write, and execute permission. Permission for a directory is often `r-x`, which means read and search. If a plus sign follows the permissions, then the file contains extended ACL entries. Use the `getfacl` command to display the ACL entries.

If you issue `ls –E`, it displays extended attributes for regular files. An additional four characters follow the original 10 characters:

```
total 11
-rwxr-xr-x+ -ps-        1 ROOT   SYS1  101 Mar 12 19:32 her
-rwxrwxrwx  a-s-        1 ROOT   SYS1  654 Mar 12 19:32 test
-rwxr-xr-x  a---        1 ROOT   SYS1   40 Mar 12 19:32 temp
-rwxr--r--  ap-l        1 ROOT   SYS1  572 Mar 12 19:32 foo
-rwxr--r--  --sl        1 ROOT   SYS1  640 Mar 12 19:33 abc
```

a
The program runs APF-authorized if linked AC=1.

p
The program is considered program controlled.

s
The program is enabled to run in a shared address space.

l
The program is considered a system-shared library object

—
The extended attribute is not set.

Creating a sticky bit file or external link for an MVS APF-authorized program

If there is a need from a z/OS UNIX environment to invoke an MVS program that is link-edited AC=1 and in an APF-authorized library, a sticky bit file or external link can be set up to point to this program. Ensure that the sticky bit file is installed with an owning UID of 0 or with the APF extended attribute, or that the external link is installed with an owning UID of 0. Because a file system mounted as NOSECURITY is considered untrusted, any file or link that is installed in a file system that is mounted as NOSECURITY is not considered trusted for this type of invocation. Also, a file with the APF extended attribute is not honored if found in a file system that is mounted as NOSETUID. Not following this setup will cause the execution of the program to fail when invoked via the z/OS UNIX spawn, exec or attach_exec callable service.

Using access control lists (ACLs)

Use access control lists (ACLs) to control access to regular files and directories by individual user (UID) and group (GID). ACLs are used in conjunction with permission bits. They are created, modified, and deleted using the **setfacl** shell command. To display them, use the **getfacl** shell command. You can also use the ISHELL interface to define and display ACLs.

The zFS and TFS file systems support ACLs. It is possible that other physical file systems will eventually support z/OS ACLs. Consult your file system documentation to see if ACLs are supported.

Before you can begin using ACLs, you must know what security product is being used. The ACLs are created and checked by RACF, not by the kernel or file system. If a different security product is being used, you must check their documentation to see if ACLs are supported and what rules are used when determining file access.

Notes:

1. The phrases *default ACL* and *model ACL* are used interchangeably throughout z/OS UNIX documentation. Other systems that support ACL have default ACLs that are essentially the same as the directory default ACLs in z/OS UNIX.
2. According to the X/Open UNIX 95 specification, additional access control mechanisms can only restrict the access permissions that are defined by the file permission bits. They cannot grant additional access permissions. Because z/OS ACLs can grant and restrict access, the use of ACLs is not UNIX 95-compliant.

ACLs and ACL entries

There are three kinds of ACLs:

- Access ACLs are ACLs that are used to provide protection for a file system object.
- File default ACLs are default ACLs that are inherited by files created within the parent directory. The file inherits the default ACL as its access ACL. Directories also inherit the file default ACL as their file default ACL.
- Directory default ACLs are default ACLs that are inherited by subdirectories created within the parent directory. The directory inherits the default ACL as its directory default ACL and as its access ACL.

Inheritance is the act of automatically associating an ACL with a newly created object. Administrative action is not needed. See "Working with default ACLs" on page 91 for more information.

There are two kinds of ACL entries:

- Base ACL entries are the same as permission bits (owner, group, other). You can change the permissions using `chmod` or `setfacl`. They are not physically part of the ACL although you can use `setfacl` to change them and `getfacl` to display them.
- Extended ACL entries are ACL entries for individual users or groups; like the permission bits, they are stored with the file, not in RACF profiles. Each ACL type (access, file default, directory default) can contain up to 1024 extended ACL entries. Each extended ACL entry specifies a qualifier to indicate whether the entry pertains to a user or a group, the actual UID or GID itself, and the permissions being granted or denied by this entry. The allowable permissions are read, write, and execute. As with other UNIX commands, `setfacl` allows the use of either names or numbers when referring to users and groups.

Managing ACLs

You need to be aware of the following rules when managing ACLs for files or directories.

- You must either be the file owner or have superuser authority (UID=0 or READ access to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class).
- You must activate the FSSEC class before ACLs can be used in access decisions.

In the following example, the RACF command activates the FSSEC class:

```
SETROPTS CLASSACT(FSSEC)
```

You can define ACLs prior to activating the FSSEC class. If you define default ACLs, they can be inherited by new objects when the FSSEC class is inactive. If the FSSEC class is not active, the standard POSIX permission bit checks are done, even if an access ACL exists. You can still display ACL information.

If files are deleted, ACLs are automatically deleted.

Working with access ACLs

The `getfacl` and `setfacl` commands are used to manage ACLs. Following are a few examples to help you get started. For details on these commands, and on other commands that support ACLs, see *z/OS UNIX System Services Command Reference*.

1. Permit user Joe and group Admins to the file named `/etc/inetd.conf` with read and write authority.

   ```
   setfacl -m user:joe:rw-,group:admins:rw- /etc/inetd.conf
   ```

   The `-m` option modifies ACL entries, or adds them if they do not exist.

2. Display the ACL that was created in Step “1” on page 90.

   ```
   getfacl /etc/inetd.conf
   #file:  /etc/inetd.conf
   #owner: BPXROOT
   #group: SYS1
   user::rw-
   group::r--
   other::r--
   user:JOE:rw-
   group:ADMINS:rw-
   ```

3. Perform the same operation as in Step “1” on page 90, but at the same time, set the base permission bits to prevent access by anyone other than the file owner.

   ```
   setfacl -s user::rw-,group::---,other::---,user
       user:joe:rw-,group:admins:rw- /etc/inetd.conf
   ```

   The `-s` option replaces the contents of an ACL with the entries specified on the command line. It requires that the base permissions be specified. The base permissions are specified similarly to extended ACL entries, except that there is no user or group name qualifier.

4. Delete the ACL that was created in Step “3” on page 90.

   ```
   setfacl -D a  /etc/inetd.conf
   ```

   The `-D a` option specifies that the access ACL is to be deleted. The permission bits remain as specified in Step “3” on page 90. When a file is deleted, its ACL is automatically deleted; there is no additional extra administrative effort required.

5. Take the ACL from `FileA` in the current directory, and apply it to FileB, also in the current directory.

   ```
   getfacl FileA | setfacl -S - FileB
   ```

   The shell pipes the output of `getfacl` to the input of `setfacl`. The `-S` option of `setfacl` says to replace the contents of the file's ACL with ACL entries specified within a file, and the "`-`" is a special case file name designating stdin. Thus, you can maintain a list of ACL entries within a file, and use that file as input to a `setfacl` command. You might use this ability to implement a "named ACL" for a given project, such as in Step “6” on page 90.

6. The file `/u/joeadmn/Admins` contains a list of ACL entries for users and groups who need to support some administrative work. The file contains ACL entries, one per line, in the format that `setfacl`

expects and which `getfacl` displays. These people must be granted access to all of the directories within the file system subtree starting and including `/admin/work`.

```
setfacl -S /u/joeadmn/Admins $(find /admin/work -type d)
```

This example uses shell command substitution to use the output of the `find` command as input to the `setfacl` command. The `/u/joeadmn/Admins` file might, for example, contain:

```
user::rwx
group::---
other::---
u:user1:rwx
u:user2:rwx
g:group1:rwx
```

7. Give Lucy read and write access to every file within Fred's home directory for which Ricky has read and write access.

```
setfacl -m user:lucy:rw- $(find ~fred -acl_entry user:ricky:+rw)
```

You can use the `find` command to search for various ACL criteria. In this example, it is used to find files containing ACL entries for Ricky, in which Ricky has at least read and write access.

You can use an access ACL on the parent directory to grant search access only to those users and groups who should have file access. The access ACL of the parent directory can have been automatically created as the result of a directory default ACL on its parent. Make sure that the 'other' and perhaps the 'group' search permission bit is off for the parent directory.

When creating ACLs, consider the following guidelines:

- To minimize the impact to performance, keep ACLs as small as possible, and permit groups to files instead of individual users. The pathlength of the access check will increase with the size of an ACL, but will be smaller than the associated checking would be for a RACF profile with the same number of entries in its access list.
- Do not disable ACLs after you have used ACLs for a while and have created many entries. Only consider disabling ACLs if you have not used them very long. If you have been using ACLs to grant, rather than deny, access to particular users and groups, then disabling ACLs will likely result in a loss of file access authority rather than a gain.

Working with default ACLs

To facilitate management of ACLs, you can define a default ACL in a directory; it will then be automatically inherited by an object.

- The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL.
- The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL. You can modify or delete inherited ACLs later.

Default ACLs have the same format as access ACLs.

Following are examples of working with default ACLS:

1. Define a directory default ACL for the directory named `/u/ProjectX`.

```
setfacl -m default:group:admins:r-x,default:group:dirgrp:rwx /u/ProjectX
```

The entries contain an extra qualifier to designate the directory default ACL. The groups named `admins` and `dirgrp` will automatically get access to any new subdirectories created within `/u/ProjectX`. Creating a default ACL will not grant access to directories that already exist.

2. Display the directory default ACL created in Step "1" on page 91.

```
getfacl -d  /u/ProjectX
#file:  /u/ProjectX
#owner: TCPAUTO
#group: SYS1
default:group:ADMINS:r-x
default:group:DIRGRP:rwx
```

   The `-d` option says to display only the extended ACL entries in the directory default ACL.

3. Define a file default ACL for the directory named `/u/ProjectX`, and all of its subdirectories.

```
setfacl -m fdefault:group:admins:r--, \
    fdefault:group:dirgrp:rw- $(find /u/ProjectX -type d)
```

   The extra entry qualifier in this case designates the file default ACL. The groups named `admins` and `dirgrp` will automatically get access to any new files created within the `/u/ProjectX` subtree. Creating a default ACL will not grant access to files that already exist.

4. Display the contents of all of the ACL types for the directory named `/u/ProjectX`.

```
getfacl -adf  /u/ProjectX
#file:  /u/ProjectX
#owner: TCPAUTO
#group: SYS1
user::rwx
group::r-x
other::r-x
user:JOE:--x
fdefault:group:ADMINS:r--
fdefault:group:DIRGRP:rwx
default:group:ADMINS:r-x
default:group:DIRGRP:rwx
```

   This example requests the access ACL (the a option), the directory default ACL (the `d` option), and the file default ACL (the `f` option). The base permission bits are displayed when the a option is specified (or defaulted).

Analyze your file system space utilization before implementing default ACLs in your file system. If you use both file and directory default ACLs in every directory in the file system, a separate physical ACL is created for every new file and directory. Using an access ACL for every directory will probably not cause concerns about space utilization. However, the same cannot be said of files, especially if the inherited ACLs are large.

Note: ACLs are not inherited across mount points. Suppose that you have a default ACL defined on the directory `/dir1/dir2`. You decide to create another directory, `/dir1/dir2/dir3`, and use it as a mount point on which to mount another file system. However, if you do so, the root directory of the mounted file system will not inherit the default ACL which had been established at `/dir1/dir2`. If you want the default ACLs of `dir2` to apply to `dir3`, you must copy them to `dir3` after `dir3` has been mounted.

Summary of tasks and their associated commands

Table 15 on page 92 summarizes the tasks that you might want to do and their associated commands.

Table 15. ACL tasks and their associated commands. This table lists each task and the associated shell command.

Task	Shell command
Add, delete, or update an ACL	`setfacl`
Display contents of an ACL	`getfacl`
Update permission bits	`setfacl` or `chmod`
Display permission bits	`ls` or `getfacl`

Table 15. ACL tasks and their associated commands. This table lists each task and the associated shell command. *(continued)*	
Task	**Shell command**
Find out whether files have extended ACL entries	`ls`
Search for files or directories that have various ACL properties	`find`
Determine if the file system and security product support ACLs	`df`
Determine if the file system supports ACLs (_PC_ACL) and also determine the maximum number of ACL entries that the file system can support (_PC_ACL_ENTRIES_MAX)	`getconf`
Restore ACL information or store the information in an archive	`pax` or `tar`
Preserve the ACLs for files and directories	The `-p` option for `cp` and the `-Z` option for `mv`.
Test files and directories for extended ACL information. Also test for directory ACLs and file default ACLs on directories.	`filetest`, `test`, [...] and [[...]] reserved-word command

How ACLs are used in file access checks

The algorithm for access checking is up to the security product that is being used. If the physical file system supports ACLs, then it uses the ck_access (IRRSKA00) callable service when passing the ACL to the security product.

If the security product supports ACLs, it applies its own rules to the file access request. RACF uses the permission bits, access ACL, and various UNIXPRIV class profiles to determine whether the user is authorized to access the file with the requested access level. For information about how RACF uses ACLs when enforcing file security, see protecting file system resources in *z/OS Security Server RACF Security Administrator's Guide*.

Auditing changes to ACLs

You can audit the creation, alteration, and deletion of ACLs by using SETROPTS LOGOPTIONS for the FSSEC class. The FSSEC class controls auditing for changes to all file security information, including file owner, permission bits, and auditing options. For more information, see Auditing for z/OS UNIX System Services in *z/OS Security Server RACF Auditor's Guide*.

Using security labels

Traditionally, access to z/OS UNIX resources is based on POSIX permissions. With the SECLABEL class active, authorization checks are performed for security labels in addition to POSIX permissions, to provide additional security. Security labels are used to maintain multiple levels of security within a system. By assigning a security label to a resource, the security administrator can prevent the movement of data from one level of security to another within the z/OS UNIX environment.

Setting security labels on z/OS UNIX

When the SECLABEL class is active, security labels can be set on z/OS UNIX resources in the following ways:

- When a physical file system or zFS aggregate is created, the file system root will be assigned the security label that is specified in the RACF data set profile that covers the data set name. If a security

label is not specified or if a data set profile does not exist, then a security label will not be assigned to the file system root.

- zFS file systems support the `chlabel` command which allows the setting of an initial security label on a file or directory. Use this command to set security labels on zFS files and directories after they have been created.
- If a directory has been assigned a security label, then new files and directories created within that directory will inherit a security label as follows:
  - If the parent directory is assigned a security label of SYSMULTI, the new file or directory is assigned the security label of the user. If the user has no security label, no label is assigned to the new object.
  - If the parent directory is assigned a security label other than SYSMULTI, the new file or directory is assigned the same security label as the parent directory.
- The rules for assigning security labels are more extensive when running in a multilevel-secure environment. For more information, see Assigning security labels to data sets in *z/OS Planning for Multilevel Security and the Common Criteria*.

Symbolic link restrictions

When security labels are used, z/OS UNIX restricts the use of certain character strings within symbolic links to allow for dynamic substitution of a user's security label within a path name.

The character strings $SYSSECA/ and $SYSSECR/ have special meaning to z/OS UNIX when they appear as the first nine characters in a symbolic link. For more information about the use of these strings, see Assigning a home directory and initial program, depending on security label in *z/OS Planning for Multilevel Security and the Common Criteria*.

Using multilevel security

Multilevel security is a security policy that allows the classification of data and users on a system of hierarchical security levels combined with a system of non-hierarchical security categories.

In a multilevel-secure z/OS UNIX environment, security labels are used as described in "Using security labels" on page 93. To set the security label on z/OS UNIX files and directories, use the `chlabel` command.

Restriction: Use DFDSS to back up and restore files while maintaining security labels. You cannot use the `pax` and `tar` commands.

Security labels for zFS files and directories

The zFS file system is the only physical file system with support for security labels in a multilevel-secure environment. For more information about multilevel security, see Assigning a home directory and initial program, depending on security label in *z/OS Planning for Multilevel Security and the Common Criteria*.

Auditing access to files and directories

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to kernel resources. An SMF record can be written at each point where the system makes security decisions.

Six classes are used to control auditing of security events. These classes do not have any profiles. They do not have to be active to control auditing. Use the SETROPTS command to specify the auditing options for the classes. For a list of the classes used for auditing and an explanation of how to specify the audit options, see Auditing for z/OS UNIX System Services in *z/OS Security Server RACF Auditor's Guide*.

Audit records are always written for the following events:

- When a user not defined as a z/OS UNIX user tries to dub a process.

- When a user who is not a superuser or not permitted to the SUPERUSER.FILESYS.USERMOUNT RACF profile tries to mount or unmount a file system.

You cannot turn off these audit records.

You can also specify auditing at the file level in the file system. Activate this option by:

1. Specifying DEFAULT in the class LOGOPTIONS on the SETROPTS command.
2. Using the `chaudit` command to specify audit options for individual files and directories.

If you activate auditing for additional levels of file system access, you might generate excessive amounts of SMF Type 80 records.

You can also specify, in a RACF user profile, that all actions taken by the user be audited. Actions taken by superusers can be audited or not, determined by RACF commands. If you are using RACF profiles in the UNIXPRIV class to control certain superuser functions, you can use those same profiles to audit those superuser functions.

Specifying file audit options

Specify file audit options using the ISPF shell, or a `chaudit` command. The command can be used to specify either user audit options or auditor audit options. To specify user audit options, you must be a superuser or the owner of the file. To specify auditor audit options, you must have RACF AUDITOR authority.

If you have AUDITOR authority, you do not need access in the permission bits to:

- Search and read any directory in the file system
- Use the `chaudit` command to change the auditor audit options for any file in the file system

If both user and auditor audit options are set, RACF merges the options and audits all the set options.

For more information, see Setting and listing audit controls in *z/OS Security Server RACF Auditor's Guide*.

Using sanction lists

You can compile a list to contain the lists of path names and program names that are sanctioned by the installation for use by APF-authorized or program controlled calling programs. This file contains properly constructed path names and program names as defined in Path and path name in *z/OS UNIX System Services User's Guide*.

Warning: Be sure that you are familiar with the activation instructions before using sanction lists. It is possible to unintentionally activate only part of this feature.

Sanction lists contain three separate lists delineated by three keywords:

:authprogram_path
This keyword is the start of a list of directories that is only used in the execution of a hfsload (or C dlload), exec, spawn, or attach_exec from an authorized program.

:programcontrol_path
This keyword is the start of a list of directories that is only used in the execution of a hfsload (or C dlload), exec, spawn, or attach_exec from an executable that is running program controlled.

:apfprogram_name
This keyword is the start of a list of program names that are allowed to get control of APF-authorized programs as a result of an exec or spawn. These names are MVS program names.

Formatting rules for sanction lists

You cannot use symbolic links (for example, $SYSNAME) in sanction lists. They will not work.

You have to follow certain formatting rules when creating sanction lists.

- Only use absolute path names.

- Path names cannot start with /*.
- Each list element must be on a line by itself, with no comments. Lines are terminated with the newline character, as is consistent with the stepliblist and useridaliastable files. Leading blanks can be on the list element line and are ignored. Use the newline character to delimit a path name. Trailing blanks are ignored. Other white space is considered part of the path name.
- Follow standard z/OS UNIX path naming conventions.
- You must follow standard MVS program naming conventions.
- Encode the sanction list file in the IBM-1047 code page.
- You can include comment lines in the list. Each comment line must start with /* and end with */. They cannot be on the same line with any other type of line.
- Do not enclose the path names or program names in quotation marks.

The tags `:authprogram_path`, `:programcontrol_path`, and `:apfprogram_name` must be used to delineate between the different types of sanction lists.

- If there are no tags in the file, then all data in the file is ignored and you will get a parsing error. If a tag is missing, then the subsequent processing of hfsload/dlload, exec or spawn will not change, based on the tag that was missing. The effect of different sanction lists is not cumulative. Once a sanction list is parsed and accepted, the contents provide the only active lists of path names and program names for hfsloads, execs, and spawns.
- List elements (path names or program names) before a tag are ignored.
- Lines after the last valid entry line (such as a path name or a program name) are ignored.
- If an `:authprogram_path` tag is present, then all lines following it and up to the next tag are considered to be approved path names from which authorized programs can be invoked.
- If a `:programcontrol_path` is present, then all lines following it and up to the next tag are considered to be approved path names from which program controlled programs can be invoked.
- If an `:apfprogram_name` tag is present, then all lines following it and up to the next tag are considered to be approved program names that can get control APF-authorized.
- If specified, the tag must start in column 1.
- The tag names are not case-sensitive.
- The list element names (for example, the path names and program names) are case-sensitive.

If the file does not follow these formatting rules, the sanction lists might not be recognized properly and various functions relating to the attempted use of the lists might fail.

Steps for creating a sanction list

Before you begin, you need to know what directories and what programs are to be set into this file. You can partially construct this file and add path names and program names as you go along. A partially complete file can be activated and when additional entries are known, this file can be updated. A background task will automatically check this file every 15 minutes for updates and then incorporate them.

You also need to be aware that only one sanction list check is done for each program invocation. Although links in directories are supported, sanction list processing only performs one check. This check uses the path name or program name that was specified by the user.

Tip: The installation can construct the sanction lists with link path names or actual path names, or both. The decision depends on how the site would like the users to invoke the programs. For example, if the actual directory is in the sanction list instead of the directory that contains the link, and the associated program is invoked via the link, the program would not be executed. The program is only executed if the directory where the link was defined or resides is specified in the sanction list and the associated program is invoked via the link. Alternatively, both the actual directory and directory where the link resides could be placed in the sanction list. This method gives users the option of invoking the program either way.

Likewise, if only the actual directory was placed in the sanction list, the user would be forced to use actual path names and not links.

Perform the following steps to create a sanction list.

1. Create a sanction list, following the rules listed in “Formatting rules for sanction lists” on page 95. You can cut and paste the following sample.

```
 /*****************************************************************/
/*                                                               */
/*   Name: Sample authorized program list                        */
/*                                                               */
/*   Description:  Contains lists of approved directories and    */
/*                 program names from which privileged programs  */
/*                 may be invoked                                */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/* Authorized program directories                                */
/*****************************************************************/
:authprogram_path
/bin/test
/bin/test/beta

/*****************************************************************/
/* Program control directories                                   */
/*****************************************************************/
:programcontrol_path
/in/test/specials

/*****************************************************************/
/* APF authorized programs                                       */
/*****************************************************************/
:apfprogram_name
PAYOUT
```

_____

2. Give the sanction list a name.

   The path name of the sanction file should be `/etc/authfile`, in keeping with IBM's strategy to place all customized data in the `/etc` directory.

_____

When you are done, you have created a sanction list and named it. To activate it, see “Steps for activating the sanction list” on page 97.

Only users with superuser authority should be given update access to sanction lists.

Steps for activating the sanction list

Before you begin: You must know what the file name is for your sanction list. This file might or might not exist, or it might not be complete, or both. If this file exists, it must be properly constructed as described in “Formatting rules for sanction lists” on page 95 even though it might not be complete.

Perform the following steps to activate the sanction list.

1. Activate the sanction list processing by specifying a value for AUTHPGMLIST. If you do not specify a value, the sanction list will not be processed. Base your choice on your particular situation.

Table 16. Methods for activating the sanction list. This table lists the methods of activating the sanction list.

If you choose this method	Then
Use the AUTHPGMLIST statement in BPXPRMxx. The sanction list might or might not have already been set up.	Customize BPXPRMxx to include the AUTHPGMLIST parameter. For example: `AUTHPGMLIST('/etc/authfile')`

Table 16. Methods for activating the sanction list. This table lists the methods of activating the sanction list. *(continued)*

If you choose this method	Then
Use SETOMVS. **You should already have set up the sanction list. Otherwise, you will get an error message warning you that the file does not exist. The path name, however, will be set. If you issue the same command with the same file name, you will not get an error message. The DISPLAY OMVS command will show the AUTHPGMLIST parameter being set. This file name is used by the background task to check whether a sanction list has been created or updated.**	Issue the SETOMVS command. For example: `SETOMVS AUTHPGMLIST='/etc/authfile'` To turn off sanction list checking, issue: `SETOMVS AUTHPGMLIST=NONE`
A nonexistent sanction list. **Use this feature only if the sanction list must not exist before it is activated. It is possible to set the sanction list value and forget that the sanction list has not been completely set up. The system might appear to be operating with sanction list processing, but in fact it is not. The background task will routinely check for the nonexistent file, but sanctioning will not occur for spawns, execs, and so on. This sanction list file must be set up for sanctioning to occur. The background task will not warn that the sanction list does not exist.**	Use either method described in this table (customize the BPXPRMxx member of SYS1.PARMLIB or use SETOMVS).

_____

2. If the sanction list has not already been created (see "Steps for creating a sanction list" on page 96), create one now.

_____

When you are done, you have activated the sanction list. A background task will sweep in the background every 15 minutes for updates. Its only job is to check for the sanction list, and if it is there, to process it. Alternatively, if a change needs to be activated sooner, you can use SETOMVS or SET OMVS =(*xx*), where *xx* specifies which BPXPRMxx file is to be used to reset the various z/OS UNIX parameters.

Tip: You can turn off sanction list checking with the SETOMVS command:

```
SETOMVS AUTHPGMLIST=NONE
```

Notes:

1. If the sanction list was not created when the system is IPLed, you can create it later and then use the SETOMVS command to dynamically add it. Be careful because you will not get a message saying that the sanction list file does not exist, although z/OS UNIX will continue to check every 15 minutes.
2. If the sanction list was created before the system is IPLed, and there are errors, the sanction list processing is disabled.
3. If the AUTHGPGMLIST statement in the BPXPRMxx member contains a nonexistent value, you will not get an error message.

4. If the sanction list is running on the system, you will get error messages when you try to run program-controlled or APF-authorized programs that are not in the sanction list. You will have to add them to the sanction list.

Maintaining the security level of the system

After you set up a secure environment for your system, you must ensure that it stays secure.

Steps for maintaining the security level of the system

Before you begin, you need to have set up a secure environment for your system.

Perform the following steps to ensure that the system stays secure.

1. Check each program that you want to introduce into the system. Add a program only if you are certain that it will not lower the level of security.

   _____

2. For users of the system, set up rules for:
   - Sharing data in files
   - Specifying permission bits when creating files or using the chmod command or chmod() callable service

   _____

3. Require that users set the permission bits for their files to deny access to all users except themselves, as the file owners.

   _____

4. Protect all local data sets with a RACF profile that specifies UACC(NONE). Only administrators with responsibility for creating, restoring, or dumping local data sets should be permitted to this profile.

   _____

When you are done, you have taken steps to ensure that your system stays secure.

Controlling access to applications

If the APPL class for the security product is active, you can use a combination of profiles in the APPL class and the APPL operand on the RACROUTE REQUEST=VERIFY macro to determine which users are allowed to use specified applications as they enter the system. For example, if you do not want all of your users to use certain applications, you can activate the APPL class and create a profile with an access list that contains only those users who are allowed to access these applications.

When specifying a profile, you have two choices: use the OMVSAPPL application ID (APPLID) or create a customized APPLID. In some cases, OMVSAPPL is the value that is always used for the APPLID parameter.

If no customization is done, the following services specify OMVSAPPL for the APPLID value. If the APPL class is active, use of these services can be limited to those users who have access to the OMVSAPPL resource in the CLASS(APPL).

- __login
- pthread_security_np
- __passwd when there is no password or password phrase change specified
- __passwd when the calling process did not call pthread_security_np

In certain cases, if you customize the APPLID-related fields in the BPXYTHLI, you can change the value used for the APPLID parameter for these services:

- pthread_security_np
- __passwd

The following C functions allow the APPLID to be specified other than OMVSAPPL when invoking the service:

- __login _applid
- __passwd_applid
- pthread_security__applid_np

For more information about protecting applications, see Program security modes in *z/OS Security Server RACF Security Administrator's Guide*.

Restricting access to z/OS UNIX file systems

You can restrict user and group access to z/OS UNIX file systems.

- A z/OS UNIX administrator can control access to file systems at their mount points by using the `setfacl` command to create, modify, and delete ACLs for specific users and groups.
- At a higher level, the security administrator can choose to restrict access to the z/OS UNIX file system for all authorization checks that involve mount point traversal. The check is performed at every mount point crossover to see if the user or group has authority to access the file system. Only those who have been given permission to covering RACF resource profiles are eligible for access. Access to objects within the file systems are subject to the superuser, owner, permission bit, ACL, and UNIXPRIV rules. Users who are designated as RACF auditors are exempt from this restriction. This check uses the RACF FSACCESS class profile to validate the authority of users or groups who are accessing the z/OS UNIX file system, as described in “Using the FSACCESS class profile to restrict access ” on page 100.
- On a more granular level, the RACF FSEXEC class profile prevents users from executing any file in a zFS or TFS file system when an FSEXEC class profile matches the file system name and the users do not have at least UPDATE permission to that profile.

These restrictions apply:

- This additional access check using the FSACCESS class profile is only supported on zFS file systems
- For z/OS UNIX, zFS file systems that are mounted with the NOSECURITY option are not subject to this access control check.
- The root file system is excluded from this access restriction.
- A given zFs file system can be protected from the whole NFS network by not permitting the NFS Server's MVS UserID to the FSACCESS class profile for that specific zFS file system. When the NFS Server is configured with Security(SAF) or Security(SAFEXP), the NFS Client remote MVS UserID might also need to be permitted to the FSACCESS class profile to avoid unexpected failures.

Using the FSACCESS class profile to restrict access

Using the FSACCESS class profile to restrict access provides a coarse-grained control to z/OS UNIX file systems and acts as a gatekeeper to the z/OS UNIX file system. When the user is permitted to access the file system, any subsequent decisions to allow file access are based on z/OS UNIX permissions and ACLs. If a security decision is needed during access validation, the ck_access (IRRSKA00) callable service is used to determine whether they have access to the file system. RACF provides authorization checking and auditing and then returns control to the file system. For more information about the ck_access callable service, see ck_access (IRRSKA00): Check access in *z/OS Security Server RACF Callable Services*.

The basic steps to restrict access to the z/OS UNIX file system are to create a resource with the identical z/OS UNIX file system name in the FSACCESS class profile, permit selected z/OS UNIX users with UPDATE access to the resource, and then activate the FSACCESS class. When the FSACCESS class profile is active, RACF first uses the FSACCESS class profile resources to determine whether the user is authorized to access the file system. If the user is authorized to access the file system resource, then RACF uses the permission bits, access ACLs, and various UNIXPRIV class profiles to determine whether the user is authorized to access the individual file system objects with the requested access level.

Restriction: If the OMVS address space is not marked as trusted, you will need to update the profiles to ensure that the user ID has the appropriate access.

Steps for giving selected users or groups access to a z/OS UNIX file system

About this task

Perform the following steps to give selected users and groups access to the specified file system and then activate FSACCESS checking. Before you begin, you need to know which users or groups will be given access to the specified file system.

Procedure

1. Define a profile in the FSACCESS class for each z/OS UNIX file system that you want to grant permission. To define a profile for OMVS.ZFS.WEBSRV.TOOLS, for example, issue:

   ```
   RDEFINE FSACCESS OMVS.ZFS.WEBSRV.TOOLS UACC(NONE)
   ```

   In general, generic profile names for file systems are allowed for resources in the FSACCESS class.

   Tip: To control a set of file systems with similar names, define a generic profile. For example, after ensuring that generic profiles were enabled for the class, define OMVS.ZFS.WEBSRV.** as a generic profile, issue:

   ```
   SETROPTS GENERIC(FSACCESS)
   RDEFINE FSACCESS OMVS.ZFS.WEBSRV.** UACC(NONE)
   ```

   _____

2. Assign UPDATE access to the selected users or groups.

   ```
   PERMIT OMVS.ZFS.WEBSRV.TOOLS CLASS(FSACCESS) ID(USER19)
   ACCESS(UPDATE)
   ```

   _____

3. Activate the FSACCESS class profile, if it is not currently active at your installation. By default, it is inactive and is not used for authorization checking.

   ```
   SETROPTS CLASSACT(FSACCESS)
   ```

   _____

4. Activate SETROPTS RACLIST processing for the FSACCESS class, if it is not already active.

   ```
   SETROPTS RACLIST(FSACCESS)
   ```

   If SETROPTS RACLIST processing is already in effect for the FSACCESS class, you must refresh SETROPTS RACLIST processing in order for new or changed profiles in the FSACCESS class to take effect.

   ```
   SETROPTS RACLIST(FSACCESS) REFRESH
   ```

   _____

Results

When you are done, you have restricted access to the specified z/OS UNIX file system to users and groups who have been explicitly permitted to covering resource profiles.

Restricting execute access in a zFS or TFS file system

To prevent users from executing any file in a z/OS File System (zFS) or Temporary File System (TFS), you can define a profile in the FSEXEC class that matches the file system name and then authorize selected users and groups who require execute class by giving them UPDATE access. The FSEXEC-eligible users

are then subject to the usual authorization checking, which includes checking for superuser authority, ownership, permission bits, access control lists (ACLs), and UNIXPRIV permissions.

For more information about restricting execute access in a zFS or TFS file system to specified users and groups, see Restricting execute access in a zFS or TFS file system in *z/OS Security Server RACF Security Administrator's Guide*.

Setting up TCP/IP security

When setting up security for TCP/IP, the TCP/IP started task's user ID and its default group must both have an OMVS segment defined. The user ID, assigned using ICHRIN03 or the STARTED class, must have UID(0). For information about defining a z/OS UNIX user to RACF, see "Defining z/OS UNIX users to RACF" on page 51.

Other TCP/IP tasks such as `ftp` and `routed` must be assigned a RACF user ID using ICHRIN03 or the STARTED class. If `ftpd` and `routed` use a different started task user ID from the TCP/IP user ID, they must have UID(0) and HOME('/').

Selecting a security level for the system

If you run daemons and servers, you need to set up the appropriate security for them. Two levels of privileges are available for servers and daemons: UNIX level and z/OS UNIX level. In providing security, you need to understand the differences between the two levels. For discussions of the levels, see:

- "Establishing the correct level of security for daemons" on page 313
- "Establishing the correct level of security for servers" on page 343

Chapter 5. Managing the z/OS UNIX file system

The z/OS UNIX file system, like other UNIX systems, is a hierarchical file system. A brief overview of hierarchical file system concepts is provided, and then tasks involved with managing file systems are discussed.

Lists of subtasks

Subtask	Associated procedure
Mounting file systems	“Steps for mounting file systems” on page 113
Setting up the alternate sysplex root	“Steps for setting up the alternate sysplex root for the dynamic replacement of the current sysplex root” on page 115
Disabling support for the alternate root file system	“Steps for removing the alternate sysplex root support” on page 116
Dynamically replacing the sysplex root file system	“Steps for dynamically replacing the sysplex root file system with F OMVS,NEWROOT=” on page 117
Customizing the `cron`, `uucp`, and `mail` utilities	“Customizing the cron, uucp, and mail utilities” on page 124
Recovering from file system problems with the root	“Steps for recovering from file system problems with the root” on page 139

Basics of the z/OS UNIX file system

The z/OS UNIX file system, like other UNIX systems, is a hierarchical file system that consists of the root file system and all the file systems that are added to it. Files are members of a directory, and each directory is in turn a member of another directory at a higher level. The highest level of the hierarchy is the root directory. Each instance of the system contains only one root directory. Under the `/usr/lpp` directory are directories for z/OS elements and features.

A hierarchical file system consists of files, directories, and additional file systems.

- The files contain data or programs. A file that contains a load module or shell script or REXX program is called an *executable file*. Files are kept in directories.
- The directories contain files, other directories, or both. Directories are arranged hierarchically, in a structure that resembles an upside-down tree, with the root directory at the top and the branches at the bottom. The root is the first directory for the file system at the top of the tree and is designated by a slash (/).
- The additional local or remote file systems are mounted on directories of the root file system or other file systems.

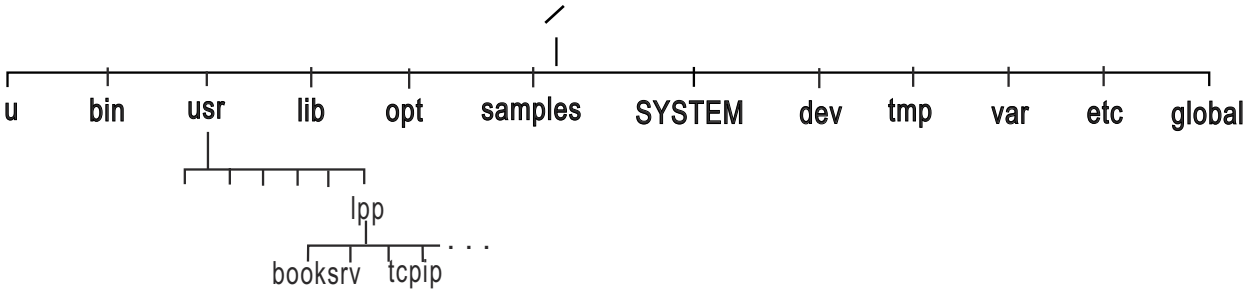


Figure 7. Logical view of the hierarchical file system for the user

Some notes to consider:

- File systems must be cataloged in the master or user catalog in order for the file systems to be mounted by z/OS UNIX.
- File systems have multiple volume support. The support allows a file system to span a total of 59 volumes with limits of 123 extents per volume and 255 extents across all volumes. The maximum size of a file that is stored in a file system is shown in the following equation.

  ```
  2**31 pages (4 K bytes/page) = 2**43 bytes
  ```

  The volume and extent limits are MVS limits. They are not file system limits.
- If you are sharing file systems in a sysplex, you know that the term "root file system" is called the *version file system*. Think of the version file system when you see "root file system".

Structure of the z/OS UNIX file system

If you follow the instructions for ServerPac and CBPDO installations, all z/OS elements and features that store into the z/OS UNIX file system are installed into a consolidated file system, instead of having separate product-related file systems. Continue this consolidated approach as you install additional products on the platform. This method makes maintaining and cloning the file system easier, and it simplifies the MOUNT statements in the BPXPRMxx member.

Important: You must maintain a separate file system for each of the following directories. When you are sharing a file system between systems, those four directories must have individual copies on each system. That is, each system should have their own copy of those four file systems mounted under those directories. You cannot share them between systems.

- `/etc`, which contains customization data. Keeping the `/etc` file system in a file system separate from other file systems allows you to separate your customization data from IBM's service updates. It also makes migrating to another release easier. As described in “Establishing an /etc file system for a new release” on page 12, after you complete instructions for a ServerPac or CBPDO installation, you will have an `/etc` file system in its own file system.
- `/dev`, which contains character special files that are used when logging into the shell environment and also during c89 processing. It is shipped empty. The necessary files are created when the system is IPLed, and on a per-demand basis.
- `/tmp`, which contains temporary data that are used by products and applications. `/tmp`, is created empty, and temporary files are created dynamically by different elements and products. You have the option of mounting a temporary file system (TFS) on `/tmp`. For more information, see Chapter 14, “Managing the temporary file system (TFS),” on page 293.
- `/var`, which contains dynamic data that is used internally by products and by elements and features of z/OS. Any files or directories that are needed are created when code is executed or customized. An example is caching data. In addition, you can be assured that IBM products will only create files under `/var` when code is executed or customized.

Command differences due to symbolic links

Certain directories such as `/etc`, `/dev`, `/tmp`, and `/var` were converted to symbolic links. Some shell commands have minor behavioral differences when referring to symbolic links than for regular files or directories. For example, **`ls`** does not follow symbolic links by default. Before Version 1 Release 9, `/etc` was a directory, so **`ls /etc`** would display all files in `/etc`. Beginning in Version 1 Release 9, `/etc` became a symbolic link, so **`ls /etc`** displays only the symbolic link. In this case, it is `/etc`.

In order to follow symbolic links, you must specify **`ls -L`** or provide a trailing slash. For example, **`ls -L /etc`** and **`ls /etc/`** both display the files in the directory that the `/etc` symbolic link points to.

Other shell commands that have differences due to symbolic links are `chmod`, `du`, `find`, `pax`, `rm`, and `tar`.

While these behavioral changes should be minor, users can tailor command defaults by creating aliases for the shell command. For example, if you want `ls` to follow symbolic links, you can issue the command:

```
alias ls="ls -L"
```

Aliases are typically defined in the user's ENV file. For more information, see alias - Display or create a command alias in *z/OS UNIX System Services Command Reference*.

After you establish the alias, **`ls`** will follow all symbolic links.

An administrator can put alias commands in the `/etc/profile` directory , which might affect login shells. This action is not suggested, because changing the default behavior in `/etc/profile` might produce unexpected results in shell scripts or by shell users.

Suggested file system structures for user directories and files

For users, you should logically mount other file systems on the root file system. Have your users place their directories and files in the mounted file systems. Separate user file systems offer several advantages:

- Managing storage is more efficient because the system administrator only needs to allocate data sets that are large enough to accommodate the needs of individual users.
- Isolating failures is easier because the system administrator can unmount the user file system that caused an error without affecting other users' data or causing z/OS UNIX to fail.
- They relieve the contention for system resources that might occur by having multiple users in a single file system.

When you work with file system structures, follow these rules:

- Name each user's home directory `/u/userid` where *userid* is the user ID in lowercase.
- Keep system file systems separate from user file systems by putting the file systems on different volumes.

For easier management of file systems, use the automount facility. It is described in Chapter 6, “Using the automount facility,” on page 147.

Using the Network File System (NFS)

A workstation user connected to a host through TCP/IP can mount all or part of a file system that is at the host so that it appears as part of the user's local file system. A combination of the TCP/IP server and Network File System (NFS) makes this possible.

If you are using the NFS server, you can make both traditional MVS data sets and z/OS UNIX files appear as part of the user's workstation file system. The user can create, delete, read, write, and otherwise treat the host-located files as an extension of the workstation's own file system. ASCII-EBCDIC conversion for

single-byte text files is performed automatically by means of default standard conversion tables. (NFS does not provide conversion of double-byte text files.)

Using the NFS client, you can access z/OS UNIX files and MVS data sets on other z/OS systems. You can also access z/OS UNIX files on any system with an NFS server and the proper protocol support.

For more information about NFS, refer to *z/OS Network File System Guide and Reference*. For more information about TCP/IP, refer to:

- *z/OS Communications Server: IP Configuration Reference*
- *z/OS Communications Server: IP Configuration Guide*

Using the z/OS File System (zFS)

zFS is a UNIX file system. It contains files and directories that can be accessed with APIs. They can also be mounted into the z/OS UNIX hierarchy along with other local or remote file systems types such as TFS and NFS. Use the ZFS file system type when installing z/OS or as the root file system.

You can use ISPF panels to create and manage the zFS file system. For more information about working in the ISPF shell, see Invoking the ISPF shell in *z/OS UNIX System Services User's Guide*.

The automount facility can be used for zFS, as described in “Automounting zFS file systems ” on page 147.

You can set up read-only basic partitioned access method (BPAM) access to zFS files. Each zFS directory is treated as if it were a PDSE or PDS directory. For more information about BPAM, see Reading UNIX files using BPAM in *z/OS DFSMS Using Data Sets*.

For more information about setting up and using zFS, see *z/OS File System Administration*.

Migrating the HFS file system to the zFS file system

As of V2R5, zFS is the primary file system and HFS is no longer supported. Prior to migrating to the V2R5 level, all existing HFS file systems should have been migrated to zFS. Once you are at the V2R5 level, you cannot mount an HFS.

Using the BPXWH2Z tool

The BPXWH2Z tool is an ISPF-based tool that can be used to migrate HFS file systems to zFS file systems. With it, you can change the space allocation, placement, SMS classes, and data set names. Invoke it from the ISPF command panel. See *z/OS Upgrade Workflow* for more information about the migration action.

The HFS must be unmounted before you can begin migrating the HFS file system to zFS.

Requirement: The zFS address space must be successfully configured and initialized before you can use the BPXWH2Z tool. For more information about the customization and initialization of zFS, see zFS installation and configuration steps in *z/OS File System Administration*.

To summarize, you can accomplish the following tasks with BPXWH2Z:

- Migrate HFS file systems (both mounted and unmounted) to zFS file systems. If the HFS being migrated is mounted, BPZWH2Z automatically unmounts it and then mounts the new zFS file system on its current mount point.
- Define zFS aggregates by default to be approximately the same size as the HFS file system. The new allocation size can also be increased or decreased.
- Have the migration run in TSO foreground or UNIX background.

You can use the JCL sample ISPBTCH in SYS1.SAMPLIB to invoke BPXWH2Z as an ISPF batch job. Read the notes section before running the job. If you manually migrate from an HFS to a zFS file system without using the BPXWH2Z tool, you must allocate and format the target zFS file systems.

Using the bpxwmigf command

You can use the **bpxwmigf** shell command to migrate HFS file systems to zFS file systems. The HFS file system does not have to be unmounted first. **bpxwmigf** is also available as a TSO/E command or REXX system command. For more information, see bpxwmigf - Migrate file systems to zFS in *z/OS UNIX System Services Command Reference*.

Mounting considerations for zFS

Keep in mind that the TYPE option is generic. When you specify either ZFS or HFS, the data set type is determined by the type of the data set.

- If you specify TYPE(HFS), a search is done for a data set that matches the file system name.
  - If the data set is found and it is not an HFS data set, the type is changed to ZFS.
  - If a data set is not found, the type is changed to ZFS.

  In both cases, the mount proceeds as though TYPE(ZFS) was specified.
- If you specify TYPE(ZFS) and it is an HFS data set, then the type is changed to HFS. The mount proceeds as though TYPE(HFS) was specified but it will fail because HFS is no longer supported.

If the file system type that was specified on the mount does not match the type of the file system and it is changed, the PARM parameter used by the mount is not preserved.

You can use /// as a placeholder in file system names for mount processing to substitute ZFS.

Restriction: zFS does not support the DDNAME() keyword on the BPXPRMxx ROOT and MOUNT parmlib statements. The FILESYSTEMNAME() keyword must be used instead.

Determining the zFS file system owner

Beginning with z/OS V1R11, the zFS file system owner can be different than the z/OS UNIX file system owner when zFS is running sysplex-aware. For an explanation on how to determine the owner of a zFS file system, see Determining the file system owner in *z/OS File System Administration*. Instead of the typical commands used to determine the z/OS UNIX user (for example, **df -v**, D OMVS,F, or F BPXOINIT,FILESYS=D,ALL), the **zfsadm lsaggr** command is used.

Setting up the z/OS UNIX file system

If you are a system programmer, you will set up and manage the z/OS UNIX file system, including the following tasks:

- Allocating the root file system.
- Mounting the root file system by placing a ROOT statement in the BPXPRMxx member of SYS1.PARMLIB. During initialization, the system mounts the file systems in the ROOT statement and in all MOUNT statements in BPXPRMxx.

  You can also change the active MOUNT attributes of the root without having to reIPL by using the TSO/E MOUNT and UNMOUNT commands. However, if you have any users who are logged on or applications running, this method can be disruptive.
- Adding directories to the root file system. You can use an empty directory as a mount point for a file system that you are mounting.
- Adding MOUNT statements in BPXPRMxx for all file systems that you mount so that they are mounted whenever the system is IPLed.

  If a file system is not mounted, the user does not have access to it. The BPXPRMxx member can contain MOUNT statements for each of the file systems that you created. You can also create a REXX exec that contains multiple MOUNT statements, one for each of the file systems.

  Various methods for mounting are:

  - Using the automount facility.

- Using a TSO/E CLIST or REXX exec.
- Issuing the TSO/E MOUNT command from `/etc/rc` using the **tso** or **tsocmd** shell command. For example:

```
/bin/tso "mount filesystem(OMVS.ZFS.F96) mountpoint('/u/d96') type(zfs)
mode(read)"
```

- Running the REXX exec `/usr/sbin/mount` from `/etc/rc`. For example:

```
/usr/sbin/mount  -f  OMVS.ZFS.D96 /u/d96
```

- Using the `/sample/samples/mountx` utility.
- Using the **mount** shell command.
- Using an automation product such as Tivoli NetView for z/OS for mounting a file system.

Naming rules for file names and path names

In the JCL used for files, the file names and path names can be in lowercase letters, in uppercase and lowercase letters, or in uppercase letters. The case of letters is important.

The editor used to create the JCL must not change the file names and path names into uppercase.

Allocating a file system for the root file system

In open systems analogous to z/OS UNIX, allocation might be called *making the file system*. The root file system must be allocated by a user who has an UID of 0, indicating superuser authority. To create file systems, a security product that supports the SAF calls made during the system processing must be running.

This example shows a sample job that defines the zFS root file system. Two steps are required. The file system must first be defined and then formatted. The allocation specified in this sample does not reflect the amount of space needed for the root file system. For exact size information, consult *z/OS Program Directory* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). Later, during customization, put the specified aggregate name of the root file system in the ROOT statement in the BPXPRMxx member. In this example the owner of the file system is assigned to UID (`-owner 0`) and the file system permissions are 0755 (`-perms o755`).

```
//USERIDA JOB ,'Compatibility Mode',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//DASD0 DD DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN DD *
    DEFINE CLUSTER (NAME(OMVS.ROOT) -
           VOLUMES(PRV000) -
           LINEAR CYL(40 1) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.ROOT -compat -owner 0 -perms o755')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

This example shows a sample job that defines the root file system in an MKFS DD statement for a zFS file system. The allocation specified in this sample does not reflect the amount of space needed for the root file system. For exact size information, consult *z/OS Program Directory* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). When you specify space for the file system, you must provide a nonzero value for the directory space parameter, which is

not used. Or you can specify DSORG=PO to create a data set with partitioned organization. The DSNAME is in the ROOT statement. Later, during customization, put the DSNAME of the root file system in the ROOT statement in the BPXPRMxx member.

```
//OMVSXX    JOB
//STEP03 EXEC  PGM=IEFBR14
//MKFS   DD   DSNAME=OMVS.ROOT,
//            SPACE=(CYL,(40,1,1)),DCB=(DSORG=PO),
//            DSNTYPE=ZFS,
//            DISP=(NEW,CATLG,DELETE),
//            STORCLAS=STANDARD
```

Other MVS data sets can reside in available parts of the volume containing the file system. You can also use the ISPF shell, or the TSO/E ALLOCATE command to create a file system.

See the following topics:

- To learn more about creating zFS file systems, see Creating and managing zFS file systems using compatibility mode aggregates in *z/OS File System Administration*.
- For JOB, EXEC, and DD statements, see *z/OS MVS JCL Reference*.
- See BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference* for BPXPRMxx reference information. For guidance information, see “Customizing the BPXPRMxx member of SYS1.PARMLIB” on page 17.
- To learn more about setting up the root file system structure, see “Structure of the z/OS UNIX file system ” on page 104.

File systems can be allocated by systems other than the one on which the data set will be used, as long as the allocating system has the correct level of DFSMS. The system where the file system will be used must share the catalog with the allocating system or have a catalog entry for the same file system name.

Defining the root file system

Use the ROOT statement in the BPXPRMxx member of SYS1.PARMLIB to define which file system is the root file system. The zFS file system is the preferred file system.

Figure 4 on page 19 shows the sample BPXPRMxx member. The ROOT statement is as follows:

```
ROOT   FILESYSTEM('OMVS.ROOT')
       TYPE(ZFS)
       MODE(RDWR)
```

The root file system is the starting point for the overall hierarchical file system. It contains the root directory and any related files or subdirectories.

What happens when file systems are mounted?

After you mount a new file system for the first time, you need to change the owner and group owner. or more information about mounting new file systems, see “Mounting file systems” on page 110.

To begin with, the hierarchical file system is used to store data and organize it in a hierarchical way by using file system entries such as directories and files. These file system entries have certain attributes, such as ownership, permission bits, and access timestamps. The data and the attributes of a file are stored with the file in the file system. All file attributes are stored in a control block that is sometimes called the *inode*.

Mounting a file system creates a *binding* for the duration of the mount. The binding is between a directory that is already in the file system hierarchy, called the *mount point*, and the entry point into the file system about to be mounted, called the *root* of this file system. The mount point directory and the root are connected until unmount time. When a file system is mounted on a mount point, it overlays the contents of the mount point directory. Files, symbolic links, and subdirectories within the mount point directory are no longer accessible and are hidden until the file system is unmounted.

The following figure shows what happens when Jane's file system is mounted on directory `/u/jane`.

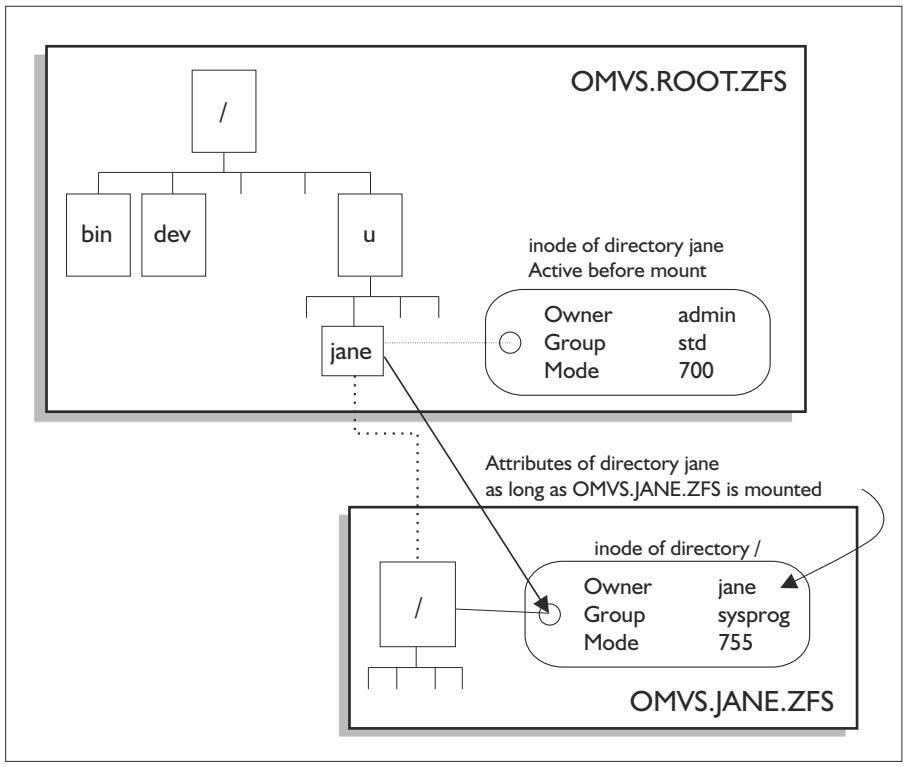


Figure 8. Mounting a file system

The root directory of a file system, like any other entry, also has attributes, but the directory does not have a name. At mount time, the mount point directory lends its name to the root directory of the file system that is to be mounted. However, the root keeps its attributes. Logically, the directory (which is an entry in another directory, one level up the hierarchical tree) no longer points to its own inode. Instead, it points to the inode of the mounted root. Thus, the attributes and the content of a directory are hidden while as a file system is mounted on it.

The root of a file system always keeps the attributes that it had at unmount time. The same attributes are used again when the file system is mounted later. The attributes do not depend on the actual mount point.

The automount facility and the ISPF shell can be used to define the zFS file system. For more information about automount: Configure the automount facility or ISHELL, see *z/OS UNIX System Services Command Reference*.

When zFS file systems are mounted

For zFS file systems, the owning user, owning group and permissions can be specified when the file system is defined. For more information, see Installing and configuring zFS in *z/OS File System Administration*.

Mounting file systems

After installation is complete, you need to create file systems and mount them to your root file system or somewhere else in the hierarchy.

If /// is used as a placeholder in file system names, mount processing will replace the /// with ZFS. If it is a HFS file system, mount processing will fail because HFS is not a supported file system type as of V2R5.

Tip: To verify that file systems specified in BPXPRMxx are mounted, use the USS_FILESYS_PARMLIB_MOUNTS check provided by IBM Health Checker for z/OS.

Security considerations when mounting

Taking the following into consideration, you should specify "setuid no" when mounting (the default is yes):

- UNIX files and directories are contained in MVS data sets.
- UNIX users who are using these files and directory do not need access to these MVS data sets.
- Only the kernel and the storage administrators need access to the data sets.

If you give the users direct access to the MVS data sets by giving them UPDATE access in a RACF profile protecting the data sets, or by naming the data sets with the user ID as the HLQ, and you do not specify "setuid no" when mounting, you will have a security exposure.

Privileged mount and unmount authority

Mounting and unmounting z/OS UNIX file systems are privileged operations. A user must have UID(0) or have read access to the SUPERUSER.FILESYS.MOUNT resource in the UNIXPRIV class before file systems can be mounted or unmounted.

The TSO MOUNT and UNMOUNT commands will perform privileged operations if the user has read access to the BPX.SUPERUSER resource in the FACILITY class.

Nonprivileged mount and unmount authority

Nonprivileged users must meet certain requirements before they can mount z/OS UNIX file systems. They must have the following access permissions:

- Read access to SUPERUSER.FILESYS.USERMOUNT profile. The SUPERUSER.FILESYS.USERMOUNT resource name in the UNIXPRIV class allows nonprivileged users to mount and unmount file systems with the `nosetuid` option.
- Read-write-execute (`rwx`) access to the directory that the file system will be mounted on.
- Read/write-execute (`rwx`) access to the file system root.

  If the file system that is being mounted is an NFS client file system or if the mount point directory is in an NFS client file system, the access check is sent to the NFS server. You must also be permitted to the directory by that server. For a z/OS NFS server, you might have to be in the server's export list or you might have to do an MVSLOGIN.

  If the file system being mounted is an NFS remote file system, or if the mount point directory is in an NFS remote file system, the access check is sent to the server for processing. In order for the access check to succeed, you must have already been permitted to the root or mount point directory, respectively, at that remote server. For an NFS server, this is controlled by the server's site security attribute. You might have to be listed in the server's export data set, or might have to issue an **`mvslogin`** command to log in to the remote z/OS NFS server, before issuing the mount command.

In addition, the directory that the file system is to be mounted on must be an empty directory. If it has the sticky bit on, you must be the owner of that directory. If the mount point directory is in a remote type file system (for example, NFS), then the owner UID of the mount point directory must match the your UID. If the file system root has the sticky bit on, you must be the owner of the root. For remote type file systems (for example, NFS), the owner UID of the file system root must match the UID of the user.

To unmount file systems, the nonprivileged user must have read access to SUPERUSER.FILESYS.USERMOUNT profile. The file to be unmounted must have been mounted by that nonprivileged user. Superusers can also unmount file systems that were mounted by nonprivileged users. The nonprivileged user must also still have access to the file system root and if the sticky bit is on, must still be the owner.

When a nonprivileged user mount fails, message BPXF084I is issued to the hardcopy log.

Use the MAXUSERMOUNTSYS and MAXUSERMOUNTUSER statements in the BPXPRMxx parmlib member to specify mount limits for nonprivileged users.

- MAXUSERMOUNTSYS is the maximum number of nonprivileged user mounts for the system or for the shared file system configuration.
- MAXUSERMOUNTUSER is the maximum number of nonprivileged user mounts for each nonprivileged user in the system or in the shared file system configuration.

The most recent specification is used for each system that is participating in a shared file system configuration. To set these values, you must specify them in the BPXPRMxx parmlib member. You can use the SETOMVS or SET OMVS command later to dynamically increase or decrease each of them. However, dynamically changing the values does not affect currently mounted file systems. If you want to use nonprivileged user mounts, you must ensure that MAXUSERMOUNTSYS and MAXUSERMOUNTUSER are both nonzero.

If a value for MAXUSERMOUNTSYS or MAXUSERMOUNTUSER is not specified in BPXPRMxx, the system uses the default value for them. For a single system, the default value is 0. For the first IPLed system in the shared file system configuration, the default value is 0. For a subsequently IPLed system in the shared file system configuration, the default value is what other systems have at the time when the subsequent system is being IPLed.

Restrictions:

1. The SYSNAME option, which specifies the name of the system to be mounted on, is not supported.
2. The use of /// as a placeholder in the file system name is not supported.
3. Nonprivileged users cannot use the `/usr/sbin/chmount` function.
4. Nonprivileged users cannot use the remount function.
5. The mount operation fails if either MAXUSERMOUNTSYS or MAXUSERMOUNTUSER is exceeded.
6. The BPX1MNT callable service is not supported for the user mount.
7. Mounting on a non-empty mount point is not allowed regardless of the NONEMPTYMOUNTPT settings
8. Errors from the security restriction are not recorded in the mount failure database. Use unique return codes and reason codes to identify the problem along with the audit failures.

Nonprivileged mount and unmount with SETUID authority

Nonprivileged users can also mount with the SETUID option if their SUPERUSER.FILESYS.USERMOUNT profile specifies UPDATE access. All other restrictions that were previously listed for nonprivileged user mounts still apply. This profile is also needed if you are unmounting the file system.

Nonprivileged container mount and unmount authority

To perform a container mount or unmount, you must have read access to the CONTAINERS profile. The CONTAINERS resource name in the UNIXPRIV class allows nonprivileged users to mount and unmount certain file systems. Also, any mount or unmount must be in an environment that is marked as a clean program-controlled environment.

Once you have the appropriate permissions, you can mount or unmount in the following situations:

- A UFS can be mounted if you have these permissions to the merged directories:
  - You are the owner of the mount point, the upper directory, and the working directory.
  - You have read and search permission to all lower directories.
- You can also unmount any UFS that you have previously mounted.
- A PROC file system can be mounted at a mount path ending in `/proc` if the mount occurs in a new mount namespace.
- A TFS can be mounted on mount paths ending in `/dev`, `/run`, `/tmp`, or mount paths starting in `/tmp/` if the mount occurs in a new mount namespace.
- You can create bind mounts if the mounts will be in a new mount namespace.

- You can unmount the old root file system after a pivot_root.

Steps for mounting file systems

Before you begin: You need to know that the mount point must be an empty directory. If it is not, then its contents will be hidden for the duration of any subsequent mounts.

Perform the following steps to mount a zFS file system.

1. Build a directory in the root file system. A directory can be used as a mount point for a file system. To build the directory, use one of the following methods:
   - The TSO/E MKDIR command interactively; in an in-stream data set in the JCL, such as SYSIN; or in a CLIST or REXX exec.
   - The **mkdir** shell command.
   - The TSO/E ISHELL command.
   - The MKDIR keyword in a ROOT or MOUNT statement in the BPXPRMxx member of SYS1.PARMLIB.

   ---

2. Allocate another zFS file system, using one of the following methods:
   - The **zfsadm define** and **zfsadm format** commands, which are described in *z/OS File System Administration*.
   - The TSO/E ISHELL command.
   - A JCL job. For more information, see Creating and managing zFS file systems using compatibility mode aggregates in *z/OS File System Administration*.

   After you have completed this step, go to Step "3" on page 113.

   ---

3. Logically mount the new file system in the directory of an existing file system by using the TSO/E MOUNT command under a user with mount authority.

   Example: The directory `/u/joe` is a mount point for OMVS.USER.JOE and `/u/jane` is a mount point for OMVS.USER.JANE.

   - If you are mounting a new zFS file system:

     ```
     MOUNT FILESYSTEM('OMVS.USER.JOE') TYPE(ZFS) MOUNTPOINT('/u/joe')
     ```

When you are done, you have mounted a file system.

Restrictions on mounting file systems

The restrictions on mounting file systems are as follows:

- The mount point must be a directory. If it is not an empty directory, files in that directory are not accessible while the file system is mounted. The BPXPRMxx parmlib statement NONEMPTYMOUNTPT can be used to control how the system mounts the file systems on the non-empty mount points. The NOWARN option specifies that the mount is to take place without any warning message when the mount point is a non-empty directory. The contents of that directory are hidden for the duration of the mount. The WARN option specifies that the mount is to take place with a warning message when the mount point is a non-empty directory. The contents of that directory are hidden for the duration of the mount. The DENY option specifies that mounting is not to take place when the mount point is a non-empty directory. During OMVS initialization, if the mount point is contained in an NFS file system, the NONEMPTYMOUNTPT setting is not honored.
- Only one file system can be mounted at a directory (mount point) at any one time.
- Systems participating in shared file system capability can mount file systems that will be shared in read/write mode.

- The file systems in the same file hierarchy cannot have the same name even if they are mounted on different mount points. This restriction remains true whether real names or alias names are specified on the FILESYSTEM operands in BPXPRMxx or on the MOUNT command. If two file systems have the same name, they cannot be mounted.
- There is a limit to the number of zFS file systems that can be mounted at one time in your system. For planning purposes, about 1 K of storage is consumed below the 16 M line for each mounted file system. You can limit the amount of storage that is consumed. To limit the amount, use the timeout capabilities of automount so that file systems are unmounted when they are not being used. This storage below the line is used for the data set allocation. If storage is not available and another data set allocation is requested, the system may be placed in a nonrestartable wait state.

  Alternatively, you can specify SWA(ABOVE) in the BPXPRMXX parmlib member to force the storage for data set allocation to be obtained above the 16 M line. For more information about the SWA parmlib statement, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.

Additional mount point considerations are as follows:

- Do not include automount-managed directories when you specify mount points in the BPXPRMxx parmlib member. Otherwise, mount failures might occur because the automount daemon will start after the parmlib mounts are processed during the IPL.
- Be careful when you add parmlib mount points that include NFS file systems within the path. A hang during IPL might occur if an NFS file system that is not locally served is mounted.

Automatically replacing the sysplex root file system with the alternate sysplex root file system if it becomes unowned

In a sysplex configuration, the *alternative sysplex root file system* is a hot standby for the sysplex root file system that is used to replace the current sysplex root file system when the sysplex root file system becomes unowned. The alternative sysplex root file system is established by using the ALTROOT statement in the BPXPRMxx parmlib member during OMVS initialization or by using the SET OMVS command.

Requirements: If you are replacing the sysplex root file system, consider the following requirements:

- A shared file system configuration is required. However, the sysplex can be a single system.
- All systems in the shared file system environment must be at z/OS V1R11 at the minimum.
- The alternative sysplex root must have the same mount points and symbolic links as the current sysplex root. The mount points are validated during processing, but the symbolic links are not. If mount points are missing, the current sysplex root is not replaced by the alternative sysplex root.
- The file system type for the alternative sysplex root and the current sysplex root must be ZFS.
- The alternative sysplex root PFS must be active on all systems in the shared file system configuration.
- If the SECLABEL class is active and the MLFSOBJ option is active, then the multilevel security label for the alternative sysplex root must be identical to the assumed multilevel security label of the current sysplex root.
- The real path name for the mount points in the current sysplex root must not exceed 64 characters in length.

Restriction: The ALTROOT statement is ignored during processing of the F BPXOINIT,FILESYS=REINIT system command. You will have to manually issue SET OMVS=(*xx*) where BPXPRMxx is the parmlib member that contains the original ALTROOT statement.

Steps for setting up the alternate sysplex root for the dynamic replacement of the current sysplex root

About this task

This topic shows how to establish an alternate sysplex root in a shared file system environment.

Before you begin, you need to ensure that the alternate sysplex root does not reside in the same volume, device, and control unit as the current sysplex root.

To minimize the single point of failure, the alternate sysplex root file system should be a different PFS type than that of the current sysplex root file system.

Procedure

1. Allocate a new file system to be used as the alternate sysplex root file system, following these rules:
   a) The UID, GID and the permission bits of the root directory in the alternate sysplex root file system must match the root directory in the current sysplex root file system
   b) If the SECLABEL class is active and the MLFSOBJ option is active, then the multilevel security label for the alternate sysplex root must be identical to the assumed multilevel security label of the current sysplex root.

   _____

2. On the alternate sysplex root, set up the mount points and the symbolic links. The mount points and the symbolic links must be same as the ones on the current sysplex root.
   a) Mount the alternate sysplex root file system at a temporary mount point (for example, `/altroot`).
   b) Select one of the following suggested ways to set up mount points and symbolic links:
      - Use the `pax` shell command to populate the alternate sysplex root file, using the current sysplex root as a source. For example:

        ```
        cd /
        pax -wr -pe -XCM ./ /altroot
        ```

      - Use `copytree` to populate the alternate sysplex root, using the current sysplex root as a source. For example:

        ```
        copytree -as / /altroot
        ```

      - Manually issue `mkdir` and `ln -s` shell commands to create the mount point directories and symbolic links similar to the current sysplex root.
   c) Unmount the alternate sysplex root.

   _____

3. Specify ALTROOT in the BPXPRMxx parmlib member with the mount point in the root directory of the current sysplex root file system.

   Restriction: The ALTROOT mount point must not exceed 64 characters in length.

   Example: To specify ALTROOT:

   ```
   ALTROOT FILESYSTEM ('OMVS.ALTROOT.ZFS')
      MOUNTPOINT('/sysalt') PARM ('FSFULL(70,10)')
   ```

   You can use the SETOMVS SYNTAXCHECK operator command to validate the ALTROOT syntax.

   _____

4. Make sure that all systems in the shared file system environment have direct access to the new file system and can locally mount it.

   _____

5. Process the ALTROOT statement by using the SET OMVS command or by initializing the OMVS with the updated BPXPRMxx parmlib member. For example:

```
SET OMVS=(xx)
```

Results

When you are done, you have established an alternate sysplex root in the shared file system configuration. The alternate sysplex root is mounted in read-only mode at the specified mount point and designated as AUTOMOVE. When the alternate sysplex root becomes the current sysplex root, it is mounted in read-only mode and designated as AUTOMOVE regardless of the current sysplex root settings.

Requirement: If you make changes to the current sysplex root after the alternate sysplex root was established, you must make the same changes to the alternate sysplex root as well.

Steps for removing the alternate sysplex root support

About this task

This topic shows how to remove support for the alternate sysplex root.

Perform the following steps to remove the alternate sysplex root support.

Procedure

1. In the BPXPRMxx parmlib member, replace the ALTROOT FILESYSTEM statement with the following statement:

   ```
   ALTROOT NONE
   ```

   Because the ALTROOT NONE and ALTROOT FILESYSTEM statements are mutually exclusive, only one can be specified in the BPXPRMxx parmlib member.

   If concatenating parmlib members result in multiple ALTROOT statements, then the first parmlib member specified on the OMVS= operator command that contains the ALTROOT statement will take effect.

   ---

2. Issue a SET OMVS operator command to process the ALTROOT NONE statement. For example:

   ```
   SET OMVS=(XX)
   ```

Results

When you are done, you have removed the alternate sysplex root support and deleted any outstanding BPXF253E messages. The alternate sysplex root file system can be left mounted as a regular file system on all systems in the sysplex. If you need to reestablish the alternate sysplex root support with the same file system name, the file system will have to be unmounted globally before it can be used in the ALTROOT FILESYSTEM statement.

Use your preferred unmount method to unmount the alternate sysplex root.

Dynamically replacing the sysplex root file system

You can dynamically replace the sysplex root file system, either with F OMVS,NEWROOT= or with the **bpxwmigf** command. **bpxwmigf** is also available as a TSO/E command or REXX system command. For

more information, see bpxwmigf - Migrate file systems to zFS in *z/OS UNIX System Services Command Reference*.

Steps for dynamically replacing the sysplex root file system with F OMVS,NEWROOT=

About this task

Before you begin: You need to ensure that the following requirements were met:

- All systems in the sysplex are at the V1R10 or later level.
- The current sysplex root PFS, and the new sysplex root PFS, are up in all the systems in shared file system configuration.

Also, be aware of the following restrictions:

- The sysplex root must be locally mounted on all systems in the shared file system configuration.
- Byte range locks must not be held on the sysplex root during replacement processing.
- The current sysplex root and the new sysplex root must be zFS.

Note the following facts:

- Remote NFS mounts of the sysplex root or any directories on it are considered active use of the current sysplex root file system.
- During the replacement, the new sysplex root must not be mounted or in use.
- Mount parameters are preserved during the replacement of the sysplex root of the same file system type (PFS). They are dropped if the file system type is different.
- Directories, data, files, and links are not copied from one file system to another.

Perform the replacement as follows:

1. To verify that the sysplex root is locally mounted on all systems, issue:

   ```
   Ro *all, D OMVS,F,NAME=root_file_system_name
   ```

   You should see CLIENT=N for each system.

   _____

2. Allocate a new file system to be used as the new sysplex root file system.

   When you allocate new file systems, follow these rules:

   - The UID, GID, and the permission bits of the root directory in the new sysplex root file system must match those of the root directory in the current sysplex root file system.
   - If the SECLABEL class is active and the MLFSOBJ option is active, then the multilevel security label for the new sysplex root must be identical to the assumed multilevel security label of the current sysplex root.

   _____

3. On the new sysplex root, set up the mount points and the symbolic links. The mount points and the symbolic links must be same as the ones on the current sysplex root.

   a. Mount the new sysplex root file system at a temporary mount point (for example, `/newroot`).

   b. Verify that all systems in the shared file system configuration have direct access to the new sysplex root file system and can locally mount it. Issue

      ```
      Ro *all, D OMVS,F,NAME=new_sysplex_root_file_system_name
      ```

      and verify that CLIENT=N for each system.

   c. Select one of the following suggested ways to set up mount points and symbolic links:

- Use the **pax** shell command to populate the new sysplex root file, with the current sysplex root as a source. For example:

```
cd /
pax -wr -pe -XCM ./ /newroot
```

- Use **copytree** to populate the new sysplex root, with the current sysplex root as a source. For example:

```
copytree -as / /altroot
```

- Manually issue **mkdir** and **ln -s** shell commands to create the mount point directories and symbolic links similar to the current sysplex root.

d. Unmount the new sysplex root.

---

4. On any system in the shared file system configuration, issue:

```
F OMVS,NEWROOT=new.root.file.system.name,COND=<YES|NO|FORCE>
```

YES
Proceed conditionally. The system checks for active usage in the current sysplex root file system and reports the active usage in a BPXF245I message. If file activity is found, the command fails with EBUSY return code and JrActivityFound reason code. Message BPXF244E is also displayed. If file activity is not found, the command continues processing to replace the sysplex root. YES is the default.

NO
Proceed unconditionally. The system checks for active usage in the current sysplex root and reports the active usage in a BPXF245I message. Replacement of the sysplex root continues.

FORCE
Forces the replacement of the current sysplex root with the new sysplex root. This option allows user to replace a failing sysplex root with the user-specified new sysplex root.

A BPXI085D WTOR message is issued to the console to confirm the FORCE option. Mount points are validated. Symbolic links are not validated.

In addition to the restrictions listed "Steps for dynamically replacing the sysplex root file system with F OMVS,NEWROOT=" on page 117, these restrictions must be met in order to use the FORCE option:

- All systems in the sysplex must be at the V1R11 or later level.
- The real path name for the mount points in the current sysplex root must not exceed 64 characters in length.

The replacement of the sysplex root file system begins. During the replacement, active connections to files and directories in the current sysplex root file system are broken.

After the replacement completes:

- The root (/) is updated on all systems in the sysplex to point to the new sysplex root file system.
- New opens go to the new sysplex root file system. The current sysplex root for the root directory is replaced for all processes in all systems. The current directory for root directory is replaced for any processes that use it.
- Old connections in the previous sysplex root file system might get EIO errors.

---

5. Update the TYPE parameter and name of the sysplex root file system in the BPXPRM*xx* member of SYS1.PARMLIB.

When you are done, you have dynamically replaced the sysplex root file system.

Managing file systems

DFSMS manages the location of all file systems on volumes. However, a file system can outgrow the space on its volume and need more space. Or activity in a file system can become so great that it slows response time. In these cases, the file system needs to be managed.

File systems can span volumes. As users add files and extend existing files, each data set can increase in size to a maximum of 123 extents if secondary extents are specified in the allocation. The system programmer can:

- Remove other data sets from the volume on which the full volume condition resides.
- Move individual UNIX files and subtrees to other volumes.
- Move the entire full file system to another file system.

The storage administrator or system programmer can monitor the space in a file system by mounting a file system with the FSFULL parameter. For example, you would see message IGW023A when the file system is 70 percent full. Then it would issue another message when the file system is 80 and 90 percent full:

```
mount parm('FSFULL(70,10)')
```

You can set up read-only basic partitioned access method (BPAM) access to UNIX files, including zFS, NFS, and TFS files. Each z/OS UNIX directory is treated as if it were a PDSE or PDS directory. For more information about BPAM access to z/OS UNIX directories, see *z/OS DFSMS Using Data Sets*.

Reducing the size of the file system

If the file system becomes too big for the volume, you can try to reduce the size of the file system:

- Create a new file system on another volume and move some files from the full file system to the new file system. Mount the new file system onto the previously full file system.
- Move a subtree from the active file system into a new file system on a different volume. Mount the new file system onto the now-empty directory that was the head of the subtree. Accesses are divided between two volumes.

  Moving a subtree, rather than individual files, retains the hierarchical structure of the file system.

Increasing the size of the zFS file system

Use the **zfsadm grow** command to extend the size of an aggregate. If the aggregate can no longer be extended, a larger aggregate can be allocated and replaced without disruption by using the **bpxwmigf** command. For more information about the **bpxwmigf** command, see bpxwmigf - Migrate file systems to zFS in *z/OS UNIX System Services Command Reference*.

The Hot Topics article Managing zFS File Systems with the BPXWMIGF Migration Command (www.ibm.com/docs/zos-hot-topics?topic=managing-zfs-file-systems-bpxwmigf-migration-command) provides additional information.

For more information about **zfsadm grow**, see zfsadm grow in *z/OS File System Administration*.

Removing unnecessary files from directories

You can use the `skulker` shell script to remove files whose access times are older than a specified number of days from any directory. It can be run manually or invoked automatically using `cron`.

The `skulker` shell script, which is located in `/samples`, should be copied. You can modify it to suit your particular needs. Possible locations for the script include `/bin` or `/usr/sbin`, especially if `skulker` is to be run from a UID(0) program. If `skulker` is to be run by users, a locally created directory called, `/usr/bin` is a possibility, but ensure that the sticky bit is on in that directory.

For more information, see skulker - Remove old files from a directory in *z/OS UNIX System Services Command Reference*.

Improving accesses to file systems

If activity for a file system becomes so extensive that accesses are slow, take one of the following actions:

- Move the file system to a volume chosen for speed because it has, for instance, a faster channel or buffered controller.
- Move a subtree from the active file system into a new file system on a different volume. Mount the new file system onto the now-empty directory that was the head of the subtree. Accesses are divided between two volumes.

  Moving a subtree, rather than individual files, retains the structure of the file system.

Unmounting file systems

To unmount all active file systems, issue the following operator command:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

It unmounts the file systems on the system that the command was issued from.

Mounting the root file system for execution

This topic describes the mounting of the root file system.The root file system contains system code and binaries, including the `/bin`, `/usr`, `/lib`, `/opt`, and `/samples` directories. These directories contain files that are installed and serviced by SMP/E.

For systems with shared file system support, mounting the root file system read-only is optional. For more information about the version file system, which is also known as the root file system, see Chapter 7, “Sharing file systems in a sysplex,” on page 155.

Restriction: In a sysplex, the `/etc`, `/dev`, `/tmp`, and `/var` directories must have their own file system, separate from the root file system. Having those files in their own file system is also good practice for non-sysplex systems.

To ensure the integrity of file systems, users must configure their systems to propagate SYSTEM ENQs on file systems. The serialization used by file system mount is a SYSTEMS ENQ on SYSZDSN and the data set name. If this ENQ is not properly maintained, the file system will be damaged. The SYSZDSN ENQ is held on the file system until the file is unmounted. A file system can be mounted read/write on a single system, or read on multiple systems. A file system that is mounted read/write cannot be shared by multiple systems.

Deciding how to mount your root

This topic helps you decide whether to keep your root file system read/write or change it to read-only for execution. Table 17 on page 120 describes the benefits and drawbacks of the two mount modes for the production system's root file system.

Table 17. Comparing read-only and read/write mode for the root file system of the execution system. This table compares the modes for the root file system of the execution system.

Mount mode	Benefits	Drawbacks
Read/write	• You can create directories or files dynamically in the root file system. • You do not have to perform the actions listed in Table 18 on page 121.	• Poorer performance for SYSPLEX file system operations because I/O must be directed between system images in a sysplex. • Someone might modify files or directories inadvertently.

Table 17. Comparing read-only and read/write mode for the root file system of the execution system. This table compares the modes for the root file system of the execution system. *(continued)*		
Mount mode	**Benefits**	**Drawbacks**
Read-only	• Better performance for sysplex file system operation because I/O will not need to be directed between system images in a sysplex. • No one can modify directories or files within the root file system inadvertently.	• No one can create new directories or files dynamically in the root file system. • You must perform the actions listed in Table 18 on page 121. • You have extra tasks related to leaving some directories in read/write mode such as `/dev`, `/tmp`, `/etc`, and `/var`, when these do not have their own file system separate from the root file system).

To decide whether you should leave the root file system read/write or change it to read-only, use the information in Table 17 on page 120, and any other information that you might have. You should consider mounting the root file system in read-only mode, especially if you are mounting the root file system in a shared file system environment.

Leaving the root file system mounted in read/write mode

To leave the root file system mounted in read/write mode, make sure that the MODE parameter of the BPXPRMxx member has been specified RDWR. For example:

```
ROOT FILESYSTEM('OMVS.ZFS.ROOT')     /* Root file system       */
         TYPE(ZFS)                   /* File system type ZFS   */
         MODE(RDWR)                  /* Mounted for read/write */
```

Post-installation actions for mounting the root file system in read-only mode

If you want to mount the root file system in read-only mode, look at Table 18 on page 121 to see if you must take any actions. The table includes all z/OS base and optional elements that install into the file system.

These actions can be taken in any order, and do not need to be performed in a certain sequence.

Table 18. Required post-installation activities for mounting a root file system in read-only mode. List of required actions when mounting the root file system in read-only mode for elements or functions	
Element or function	**Required action**
Common Information Model (CIM)	No required actions.
Communications Server	No required actions.
Cryptographic Services Open Cryptographic Services Facility (OCSF)	No required actions.
Cryptographic Services PKI Services	No required actions
Cryptographic Services System Secure Sockets Layer Programming	No required actions.
DFSMSdfp, DFSMSdss, DFSMShsm, DFSMSrmm, and DFSMStvs	No required actions.

Table 18. Required post-installation activities for mounting a root file system in read-only mode. List of required actions when mounting the root file system in read-only mode for elements or functions *(continued)*	
Element or function	**Required action**
Hardware Configuration Definition (HCD)	No required actions.
Hardware Configuration Manager (HCM)	No required actions
IBM HTTP Server Powered by Domino	No required actions.
IBM Tivoli Directory Server	No required actions.
Infoprint Server	Change ownership of the Infoprint Server files to the Infoprint Server GID by running the `aopsetup` customization script. Also, a separate file system must be mounted read/write on the `/var` mount point. See *z/OS Program Directory* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information.
Integrated Security Services Open Cryptographic Enhanced Plug-ins (OCEP)	No required actions.
Integrated Security Services - Enterprise Identity Mapping (EIM) and Network Authentication Service	No required actions.
Language Environment	No required actions.
Library Server	See the topic on advanced customization parameters in *z/OS Program Directory* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
Metal C Runtime Library	No required actions.
Network File System (NFS)	No required actions.
Runtime Library Extensions	No required actions.
SMP/E	No required actions.
XL C/C++	No required actions.
z/OS Font Collection	No required actions.
z/OS UNIX System Services	Post-installation activities might be necessary for some installations. To determine whether actions are required, see "Customizing the cron, uucp, and mail utilities for a read-only root file system" on page 123. The actions include moving the files that are associated with the `cron`, `uucp` and `mail` utilities from the root file system.

Mounting the root file system in read-only mode

After you perform the actions in Table 18 on page 121, update the BPXPRMxx member of SYS1.PARMLIB as follows:

```
ROOT FILESYSTEM('OMVS.ROOT')         /* Root file system        */
         TYPE(ZFS)                   /* File system type ZFS    */
         MODE(READ)                  /* Mounted for read        */
```

With the root file system mounted in read-only mode, you must define other BPXPRMxx parameters for the directories that remain read/write.

Example: You might also have the following BPXPRMxx entry for /etc:

```
MOUNT FILESYSTEM('OMVS.ETC')         /* The /etc file system         */
      MOUNTPOINT('/etc')             /* Mount at /etc file system    */
         TYPE(ZFS)                   /* File system type ZFS         */
         MODE(RDWR)                  /* Mounted for read/write       */
```

Customizing the cron, uucp, and mail utilities for a read-only root file system

As of z/OS V1R13, ServerPac is delivered with the /usr/lib/cron, /usr/mail, and /usr/spool directories as symbolic links. The required directories and symbolic link structure are created during installation by the BPXMKDIR REXX exec in SYS1.SAMPLIB. The exec is invoked by BPXISMKD and can also be invoked at other times as needed.

For systems without shared file support, symbolic links must be created for the **cron**, **uucp**, and **mail** utilities before the root file system can be mounted in read-only mode. Files that were written to by those utilities must be moved out of the root file system and into a directory in a file system that was mounted in read/write mode.

Typically, no action is needed in order for the BPXMKDIR exec to set up the required directories and symbolic links. However, if files have been generated by a previous customization of the **cron**, **uucp**, and **mail** utilities, they must be moved to the appropriate /var directories before BPXMKDIR can create the needed files and symbolic links. The procedure is described in "Customizing the cron, uucp, and mail utilities" on page 124. In order to retain the customization, they should be moved to the directories that the symbolic links will point to.

The symbolic links are directed to /var directories as follows:

File	Linked to
/usr/lib/cron	../../var/cron
/usr/spool	../var/spool
/usr/mail	../var/mail

For files used by **uucp**, these files are delivered as symbolic links that are directed to the /var/uucp directories as follows:

File	Linked to
/usr/lib/uucp/Systems	../../../var/uucp/System
/usr/lib/uucp/Devices	../../../var/uucp/Devices
/usr/lib/uucp/Dialers	../../../var/uucp/Dialers
/usr/lib/uucp/Dialcodes	../../../var/uucp/Dialcodes
/usr/lib/uucp/Permissions	../../../var/uucp/Permissions
/usr/lib/uucp/config	../../../var/uucp/config

Certain directories must be in the /var directory and have the appropriate permissions.

- For the **mail** utility:
  - /var/mail with permissions set to 755../../../var/uucp/config
- For the **cron** utility:
  - /var/cron with permissions set to 755
  - /var/spool with permissions set to 755

  - /var/spool/cron with permissions set to 755
  - /var/spool/cron/atjobs with permissions set to 755
  - /var/spool/cron/crontabs with permissions set to 755
- For the **uucp** utility:
  - /var/uucp with permissions set to 774
  - /var/spool with permissions set to 755
  - /var/spool/locks with permissions set to 774
  - /var/spool/uucp with permissions set to 774
  - /var/spool/uucppublic with permissions set to 1777
  - /var/spool/uucp/.Xqtdir with permissions set to 774
  - /var/spool/uucp/.Sequence with permissions set to 774
  - /var/spool/uucp/.Status with permissions set to 774

  The **uucp** files must be in the /var directory:
  - /var/uucp/Systems
  - /var/uucp/Devices
  - /var/uucp/Dialers
  - /var/uucp/Dialcodes
  - /var/uucp/Permissions
  - /var/uucp/config

Migration considerations for the cron, uucp, and mail utilities

When migrating to a new release, verify that all necessary files and directories have been created in each /var directory in each file system on all system images. If you use the /var directories and the provided symbolic links, you do not need to customize the cron, uucp, and mail utilities in order for the root file system to be mounted in read-only mode.

However, if you decide not to use the /var directory and the provided symbolic links, you can create symbolic links from the /var directory or file to your preferred directory or file. With that setup, each time you migrate to a new release, you will have to customize the *z/OS Upgrade Workflow* in order for the root file system to be mounted read-only.

Customizing the cron, uucp, and mail utilities

This topic contains steps that are needed to set up the **cron**, **uucp**, and **mail** utilities for read-only file systems. See *z/OS UNIX System Services Command Reference* for additional customization instructions for configuring these utilities.

The instructions work for most customer environments. Customers with different customized environments should use this information as a basis to make their respective changes. The instructions are based on the following assumptions:

- The target system was IPLed in your test environment, the root file system data set is mounted on / (slash = the root) in read/write mode
- The /var file system is mounted on the /var mount point in read/write mode
- File systems such as the sysplex root and the system-specific file systems are not being used or being mounted while the customization is taking place.

Steps for customizing the cron, uucp, and mail utilities

The steps assume that you have used **cron**, **uucp**, and **mail** and have not followed the recommended customization in *z/OS Upgrade Workflow*. These changes to the /var directories must be performed on every /var directory on every system image in the sysplex.

Before beginning these tasks, you must meet the following requirements:

- Have superuser authority, such as UID(0). If you have READ access to the BPX.SUPERUSER resource in the FACILITY class, you can execute setuid(0) or the **su** command to gain superuser authority. If you have permission to the corresponding UNIXPRIV class profile, you will have authority to use specific authorized services.
- Log in to the shell environment through TSO, telnet, rlogin, or OpenSSH.

First step: If you have used **cron**, **uucp**, and **mail** and have not followed the recommended customization in *z/OS Upgrade Workflow*, you must move the contents of the /usr/spool directory to the /var directory and create a symbolic link. Ensure that no file systems are mounted on the /usr/spool directory or on any mount point under this directory. If there are any, they must be unmounted.

1. Create a directory called /var/spool. Issue:

   ```
   mkdir /var/spool
   ```

2. Optional. Because spool directories tend to be used heavily, it is good practice to create a new file system and mount it on /var/spool.
3. Change the permission setting of the /var/spool directory to 755. Issue:

   ```
   chmod 755 /var/spool
   ```

4. Change the current working directory to /usr/spool. Issue:

   ```
   cd /usr/spool
   ```

5. Copy the contents of the /usr/spool directory into /var/spool. Issue:

   ```
   pax -rw -pe ./ /var/spool
   ```

   You can choose to move the contents to a directory other than /var/spool (for example: /etc/spool). If you do, you will have to create another symbolic link later.
6. Repeat steps 1-5 for every system image in your sysplex.
7. Check that the copy was successful. If the copy was successful, then remove the /usr/spool directory. Issue:

   ```
   rm -fr /usr/spool
   ```

8. Create a symbolic link for /usr/spool that points to /var/spool. Issue:

   ```
   ln -s ../var/spool /usr/spool
   ```

When you are done, you have moved the contents of the /usr/spool directory to the /var/spool directory and created a symbolic link for /usr/spool that points to /var/spool.

If you moved the contents of the /usr/spool directory to a directory other than /var/spool (for example: /etc/spool), you will have to create another symbolic link from /var/spool that points to /etc/spool. Use these steps:

1. Change the current working directory to the /var directory. Issue:

   ```
   cd /var
   ```

2. Remove the /var/spool directory after making sure that it is empty. If it is not empty, then move the items from this directory into /etc/spool. Issue:

```
rmdir /var/spool
```

3. Create a symbolic link for /var/spool that points to /etc/spool. Issue:

```
ln  -s  ../etc/spool   /var/spool
```

Second step: If you have used **cron**, **uucp**, and **mail** and have not followed the recommended customization in *z/OS Upgrade Workflow*, you must move the contents of the /usr/lib/cron directory to the /var directory and create a symbolic link. Ensure that no file systems are mounted on the /usr/lib/cron directory or on any mount point under this directory. If there are any, they must be unmounted.

1. Create a directory called /var/cron. Issue:

```
mkdir /var/cron
```

2. Change its permission setting to 755. Issue:

```
chmod 755 /var/cron
```

3. Change the current working directory to /usr/lib/cron. Issue:

```
cd /usr/lib/cron
```

4. Copy the contents of the /usr/lib/cron directory into /var/cron. Issue:

```
pax -rw -pe ./ /var/cron
```

   You can choose to move the contents to a directory other than /var/cron (for example: /etc/cron). If you do, you will have to create another symbolic link later.

5. Repeat steps 1-4 for every system image in your sysplex.
6. After you are sure that the copy was successful, you can remove the /usr/lib/cron directory. Issue:

```
rm -fr /usr/lib/cron
```

7. Create a symbolic link for /usr/lib/cron that points to /var/cron. Issue:

```
ln -s ../../var/cron /usr/lib/cron
```

When you are done, you have moved the contents of the /usr/lib/cron directory to the /var/cron directory and created a symbolic link for /usr/lib/cron that points to /var/cron.

If you chose to move the contents of the /usr/lib/cron directory to a directory other than /var/cron (for example: /etc/cron), you will need to create an additional symbolic link from /var/cron to /etc/cron. Use these steps:

1. Change the current working directory to the /var directory. Issue:

```
cd /var
```

2. Remove the /var/cron directory after making sure that it is empty. If it is not empty, then move the items from this directory into /etc/cron. Issue:

```
rmdir /var/cron
```

3. Create a symbolic link for /var/cron that points to /etc/cron. Issue:

```
ln  -s  ../etc/cron   /var/cron
```

Third step: If you have used **cron**, **uucp**, or **mail** and have not followed the recommended customization in *z/OS Upgrade Workflow*, you must move the **uucp** files and create a symbolic link.

1. Create a directory called /var/uucp. Issue:

```
mkdir /var/uucp
```

2. Change its permission setting to 774. Issue:

```
chmod 774 /var/uucp
```

3. Change the current working directory to /usr/lib/uucp. Issue:

```
cd /usr/lib/uucp
```

4. Issue **ls -al** to see if the following files are in /usr/lib/uucp.

```
Systems
Devices
Dialers
Dialcodes
Permissions
config
```

   If none of these files show up in the directory listing, then you are done.

5. If any of the files listed in Step 4 exist, copy them to the /var/uucp directory by using the **cp** command. For example, if all the files exist, then you must copy them to the /var directory. Issue:

```
cp -p Systems /var/uucp/Systems
cp -p Devices /var/uucp/Devices
cp -p Dialers /var/uucp/Dialers
cp -p Dialcodes /var/uucp/Dialcodes
cp -p Permissions /var/uucp/Permissions
cp -p config /var/uucp/config
```

   You can choose to move the contents to a directory other than /var/uucp (for example: /etc/uucp). If you do, you will have to create an additional symbolic link later.

6. Repeat Steps 1-5 for every system image in the sysplex.
7. After you are sure that the files were copied successfully, you can remove them from the /usr/lib/uucp directory. Issue:

```
rm Systems
rm Devices
rm Dialers
rm Dialcodes
rm Permissions
rm config
```

8. Create symbolic links for these files from /usr/lib/uucp to /var/uucp. Issue

```
ln -s ../../../var/uucp/Systems Systems
ln -s ../../../var/uucp/Devices Devices
ln -s ../../../var/uucp/Dialers Dialers
ln -s ../../../var/uucp/Dialcodes Dialcodes
ln -s ../../../var/uucp/Permissions Permissions
ln -s ../../../var/uucp/config config
```

If you chose to copy the files from the /usr/lib/uucp directory to a directory other than /var/uucp (for example: /etc/uucp), you must create additional symbolic links from /var/uucp to this directory. You can use these commands to perform these actions:

1. Change the current working directory to the /var directory. Issue:

```
cd /var
```

2. If the following files exist, copy them to /etc/uucp. Issue:

```
cp -p Systems /etc/uucp/Systems
cp -p Devices /etc/uucp/Devices
cp -p Dialers /etc/uucp/Dialers
cp -p Dialcodes /etc/uucp/Dialcodes
```

```
cp -p Permissions /etc/uucp/Permissions
cp -p config /etc/uucp/config
```

3. Remove the following /var/uucp files, if they exist. Issue:

```
rm Systems
rm Devices
rm Dialers
rm Dialcodes
rm Permissions
rm config
```

4. Create symbolic links for these files from /var/uucp to /etc/uucp. Issue:

```
ln -s ../../etc/uucp/Systems Systems
ln -s ../../etc/uucp/Devices Devices
ln -s ../../etc/uucp/Dialers Dialers
ln -s ../../etc/uucp/Dialcodes Dialcodes
ln -s ../../etc/uucp/Permissions Permissions
ln -s ../../etc/uucp/config config
```

Fourth step: If you have used cron, mail, or uucp and have not followed the recommended customization in *z/OS Upgrade Workflow*, you must move the contents of the /usr/mail directory to the /var directory and create a symbolic link. Ensure that no file systems are mounted on the /usr/mail directory or on any mount point under this directory. If there are any, they must be unmounted.

Before you begin, you must login to the shell environment through TSO, telnet, rlogin, or OpenSSH.

Perform the following steps to customize the **mail** utility.

1. Create a directory called /var/mail

```
mkdir /var/mail
```

2. Change its permission setting to 775.

```
chmod 775 /var/mail
```

3. Change the current working directory to /usr/mail.

```
cd /usr/mail
```

4. Copy the contents of the /usr/mail directory into /var/mail. Issue:

```
pax -rw -pe ./ /var/mail
```

   You can choose to move the contents to a directory other than /var/mail (for example: /etc/mail). If you do, you will have to create an additional symbolic link later.

5. Repeat steps 1-4 for every system image in your sysplex.
6. After you are sure that the copy was successful, you can remove the /usr/mail directory:

```
rm -fr /usr/mail
```

7. Create a symbolic link for /usr/mail that points to /var/mail. Issue:

```
ln -s ../var/mail /usr/mail
```

When you are done, you have customized the **mail** utility. If you unmounted any file systems that were mounted on or under /usr/mail, you can mount them now using the same mount point as before. That directory must contain the contents of /usr/mail.

If you chose to move the contents of the /usr/mail directory to a directory other than /var/mail (for example: /etc/mail), you will need to create an additional symbolic link from /var/mail to this directory using these instructions:

1. Change the current working directory to the /var directory. Issue:

```
cd /var
```

2. Remove the /var/mail directory after making sure that it is empty. If it is not empty, then move the items from this directory into /etc/mail.

```
rmdir /var/mail
```

3. Create a symbolic link for /var/mail that points to /etc/mail. Issue:

```
ln -s ../etc/mail  /var/mail
```

Remounting a mounted file system

To remount the file system, use the TSO/E UNMOUNT command or the ISPF shell. The REMOUNT operand on the UNMOUNT command specifies that the specified file system be remounted, changing its mount mode.

Conditions under which you would remount a mounted file system are as follows:

- Maintenance cannot be performed on a read-only file system. The file system must be unmounted and then mounted again as read/write. If there are cascaded mounts, all of the file systems mounted on top of that file system must also be unmounted. You can unmount and remount a root file system. However, if the file system is a shared read-only root file system in a sysplex, you must unmount the root on all other systems in the sysplex.
- When you are not using shared file systems, you can use the remount facility to mount file systems as read-only under normal operating situations and as read/write to perform maintenance.

If a file is opened for a write, this is not checked if a remount operation changes the file system from read/write to read-only. Subsequent writes to the file will fail.

If a problem occurs with the remount, determine the failure, correct the problem, and try the remount again. The file system might be unavailable until the problems are corrected.

Copying the file system

To copy file systems, use the DFSMSdss COPY DATASET command.

Backing up file systems

Many new applications in z/OS exploit z/OS UNIX and store data into the file system. Some customers have ported their applications over from other UNIX or NT platforms, but might not be familiar with the licensed programs available to back up those applications on the z/OS platform. Other customers have used MVS and z/OS for a while, and assume that the backup techniques that they use for their MVS data sets are adequate for z/OS UNIX files as well. Those issues are addressed and suggestions are provided to help you plan your backup strategies for your file system data.

Tip: The /dev file system contains character special files that are created on a per-demand basis. Backing up the /dev file system is not necessary because all files are created when first referenced. Avoid backing up the /dev file system because the pseudoterminal files (/dev/ttyp*) might not close correctly when the process is terminated. For example, file attributes of the subsidiary pseudoterminal might not be restored on CANCEL flows if the /dev file system is quiesced when the close() is processed. ICH408I security error messages on subsequent open attempts might be issued. Consider mounting /dev as a TFS file system, which is typically not included in backup policies.

Ways to back up file systems

There are three models for backing up applications:

- The application maintains a transaction log for each transactional unit of work. A backup can be taken any time the application is running. If the data is recovered, the transaction log can be used to either back out, or redo, the transactions to reach a known sync point for the application.
- The application provides a quiesce capability, during which time transactional updates are suspended.
- The application does not provide either a transaction log or quiesce capability. In this simplest case, it is suggested that the backups occur only when the files are closed. If backups are taken while the files are open, it might be difficult to determine which transactions were fully recorded, and which others were still in flight.

IBM offers three ways to perform backups and maintain an inventory of their attributes:

- DFSMShsm
- Tivoli Storage Manager, formerly known as ADSTAR Distributed Storage Manager (ADSM)
- DFSMSdss

DFSMShsm

Unlike other non-VSAM data sets that can be opened and closed repeatedly throughout the day, some file systems are often mounted for several days or weeks at a time, with the individual file members inside opened as needed. Normally, DFSMShsm's automatic backup (AUTOBACKUP) processes file systems at most once per mount, so a file system mounted for a week would only have one backup taken for that week. For some applications, that might not be frequent enough. Fortunately, DFSMShsm provides some alternatives to ensure that backups are taken more frequently.

- An SMS-managed storage group can be defined with guaranteed backup frequency (GBF). For example, if GBF=3 days, then if a backup has not been taken for a particular data set in the last three days, a fresh backup is taken, whether the file has been updated or not. Since this applies to all data sets on a storage group, some customers have placed their file systems into a unique storage group with a specification of GBF=1, so as not to affect other types of data.
- Backups once a day might not be frequent enough. DFSMShsm provides commands to invoke backups to be taken, independent of the standard autobackup cycle and window. The BACKVOL TOTAL command can be used to back up all the files on a single DASD volume, a list of DASD volumes, a single storage group, or a list of storage groups. This command can be invoked from a job scheduling package such as Tivoli Workload Scheduler for z/OS, or console automation package, such as Tivoli NetView for z/OS.
- If file systems are intermixed on the DASD volumes with other data set types, you might want to back up the file systems individually. You can use the DFSMShsm command BACKDS to back up a single data set, or a set of data sets that match a particular mask filter. The DFSMShsm batch program ARCINBAK can be used to back up a list of data sets that support JCL backward reference and variable substitution. DFSMShsm also provides ABACKUP, which identifies which file systems are part of a single aggregate list, and backs these up as a single entity. You can invoke both the BACKDS and ABACKUP commands from job scheduling or console automation software.
- If the application was developed in-house, you can modify it to perform the backups internally. It might be able to perform its own quiesce process, or coordinate timestamps with its own transactional log. DFSMShsm provides the ARCHBACK assembler macro interface.

If a file system is mounted for read/write to a single MVS image, back it up by DFSMShsm from the MVS image that has it mounted. For automatic backup, you might need to designate host affinity by specifying a system name associated with AUTOBACKUP for each storage group. For command-initiated backups, you might need to ensure that the commands or batch jobs are issued to the correct MVS image.

If the file system being dumped by DFSMShsm is mounted as read/write, then this file system can only be dumped from the system on which it is mounted. If the file system is mounted as read-only or is in a sysplex (mounted read-only or read/write), then it can be dumped from any system that has access to it.

If you use DFSMShsm, you must define a user ID for the DFSMShsm address space. For DFSMShsm to access the file systems, it must run under a user ID that is set up for access to a z/OS UNIX system. When you set up this access:

- The default group for the DFSMShsm user ID must have an OMVS segment defined and a group ID associated with it.
- The home directory must be the root file system.

Tivoli Storage Manager

Tivoli Storage Manager offers another way to back up file systems. You can use this program in combination with, or instead of, DFSMShsm for your backup needs related to file systems. Tivoli Storage Manager is a client/server based product and offers some additional features that are not available from DFSMShsm.

- The z/OS UNIX System Services Client is available to back up the individual files and directories within a file system.
- Tivoli Storage Manager features a Central Scheduling component which can schedule z/OS UNIX client activity, such as "Selective Backup" at defined intervals. For example, you can back up the files in a specific subdirectory every four hours.
- Separate policies can be applied to a specific client node such that individual directories and files within a given file system can be effectively managed using retention, expiration and versioning attributes. These sophisticated features of Tivoli Storage Manager used in conjunction with a comprehensive INCLUDE/EXCLUDE list on the z/OS UNIX client platform provide a great deal of control over what is backed up and how the data is managed.
- End users can recover individual files that they have appropriate authority access to.

DFSMSdss

If you use DFSMSdss to dump or restore an active file system, the user ID must be set up to have superuser authority to quiesce and unquiesce a file system. If file systems are not mounted, then it is treated as an MVS data set and the user ID must have read authority for dump purposes and update authority for restore purposes.

Creating the user file systems

A user file system is allocated in the same way as you created the root file system. Choose a data set name that has the user name as one of the qualifiers and a size that provides sufficient space for the user's requirements.

Although the file system does not have to be SMS-managed, it is still highly suggested. Multivolume file systems are only supported as SMS-managed. (That is, you cannot have multivolume non-SMS-managed data sets.) As a user adds files and extends existing files, the data set increases in size to a maximum of 123 extents if secondary extents are specified in the allocation.

If more space is required, you might want to increase the size on the allocation or you might want to create additional file systems on different DASD volumes for a user and mount them at different mount points in the user's hierarchy.

The newly allocated data set has a root whose permission bits are set at 700. You can change the permissions only after the data set is mounted. See “Changing the permission bits for a file” on page 85 for more information about changing permission bits for a file or directory.

The following example is a sample JCL to create a file system. Change the JCL where needed.

```
//USERIDA JOB ,'Compatibility Mode',
//        CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP  DD SYSOUT=H
//DASD0    DD DISP=OLD,UNIT=3390,VOL=SER=ZFSVOL
//SYSIN    DD *
     DEFINE CLUSTER (NAME(OMVS.USER1) -
            VOLUMES(ZFSVOL) -
            LINEAR CYLINDERS(10 1) SHAREOPTIONS(3))
/*
```

```
//CREATE EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.USER1 -compat -owner 11 -perms o755')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Making user file systems available

After the user's file system is allocated, you need to mount it at a mount point off the root directory to make it available. The preferred place to mount all user file systems is a user directory under the /u user directory. In the z/OS system, there are two ways to accomplish this:

1. Direct mount. For a direct mount, allocate an intermediate file system (we called it OMVS.USERS) to be mounted between the root file system and all user file systems. Create a mount point using the **mkdir** command and issue the **mount** command. (To make the mount permanent, you will need to add the file system name and its mount point to the BPXPRMxx member.) Figure 9 on page 132 shows this.

   For more information, see "Using direct mount" on page 133.

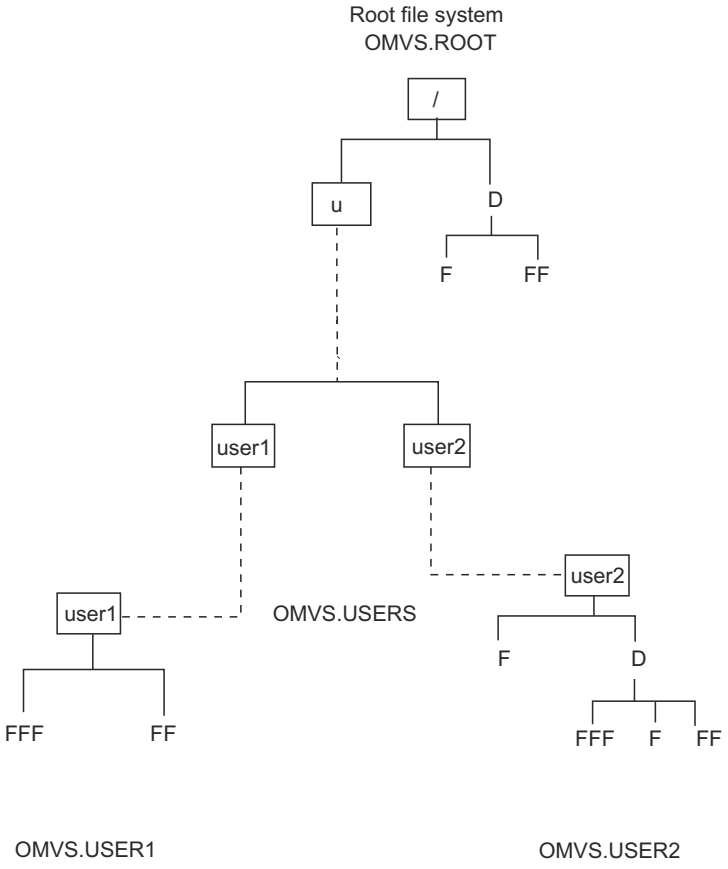


Figure 9. Using direct mount to make user file systems available

2. Automount facility. You must customize the automount facility to control all user file systems to automatically mount them when they are needed. This method is the preferred way of managing user file systems because it saves administration time. Figure 10 on page 133 shows this. See "/etc/auto.master" on page 148 for more information.

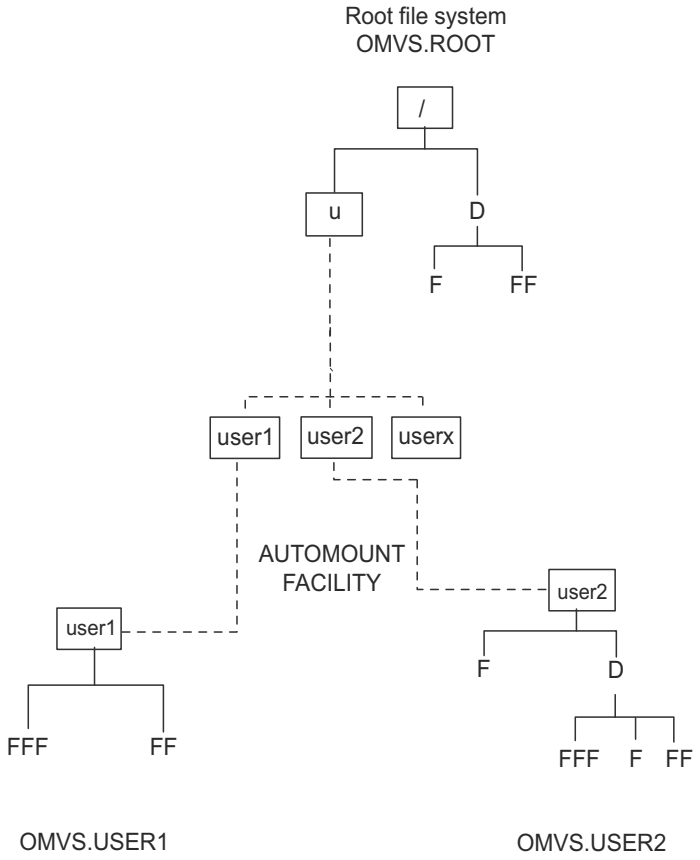


Figure 10. Using the automount facility to manage user file systems

Using direct mount

The root file system should be set up so that it does not require frequent changes or updates outside of SMP/E maintenance. To achieve this, we will allocate an intermediate file system called OMVS.USERS and mount it at /u.

All user directories that are added will reside in this new file system and not in the root file system.

Following is a sample JCL to allocate the intermediate file system. Change the JCL to fit your environment.

```
//USERIDA JOB ,'Compatibility Mode',
//        CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP  DD SYSOUT=H
//DASD0    DD DISP=OLD,UNIT=3390,VOL=SER=ZFSVOL
//SYSIN    DD *
     DEFINE CLUSTER (NAME(OMVS.USERS) -
            VOLUMES(ZFSVOL) -
            LINEAR CYLINDERS(5 1) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.USERS -compat -owner 11 -perms o755')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
...
```

The next thing to do is mount this new intermediate file system at /u. The mount can be performed from an ID that has superuser authority by:

- Using the `usr/sbin/mount` REXX exec from the shell.
- Using the TSO MOUNT command.

- Using the `mount` shell command.
- Using the ISHELL File_Systems pull-down.
- Adding an entry to the BPXPRMxx member in SYS1.PARMLIB so that it will be mounted when the system reIPLs.

An example of the commands required, including issuing the `mount` command from the shell is shown in Figure 11 on page 134. Type OMVS from ISPF option 6 to enter the shell. Then execute the highlighted commands to mount the file system OMVS.USERS. In Figure 11 on page 134, the user ID is ADMIN and it has superuser authority.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993, 1995
(C) Copyright Mortice Kern Systems, Inc., 1985, 1994
(C) Copyright Software Development Group, Univ. of Waterloo, 1989


U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - Improve performance by preventing the propagation -
 - of TSO/E or ISPF STEPLIBs                          -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# /usr/sbin/mount /u omvs.users          1
OMVS.USERS is now mounted at
/u
# df -P                          2
Filesystem     512-blocks        Used  Available  Capacity Mounted
OMVS.USERS          7200          40       7160        1% /u
OMVS.ROOT          82800       79608       3192       97% /
# chmod 755 /u                  3

 ===>

ESC=¢ 1=Help    2=SubCmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Figure 11. Mounting the new intermediate file system

- 1 Use the `mount` command to mount the file system, OMVS.USERS, on mount point `/u`.
- 2 Run the display free space command to display the mounted file systems.
- 3 Change the permission bits to allow access to `/u`.

Now that the OMVS.USERS file system is mounted at mount point `/u` you can create the user1 mount point from a superuser ID by using:

- The `mkdir` command in the shell
- The TSO/E MKDIR command
- The ISHELL Directory pull-down

Figure 12 on page 135 shows the sequence of commands performed by a superuser in the shell to create a mount point for a new user off `/u`. Before you begin, be sure that the new user is defined to the OMVS segment that your security product uses. Type in OMVS from ISPF option 6 to enter the shell and execute the highlighted commands to create the mount point for *user1*.

```
# cd /u                                1
# pwd                                  2
/u                                     3
# mkdir -m 700 user1                   4
#ls -l      5
total 16
drwx------   2 ADMIN   OMVSGRP        0 Nov  7 09:07 user1 6
#
 ===>

ESC=¢ 1=Help    2=SubCmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Figure 12. Creating a mount point directory for a user

- 1 Change to make /u your current working directory.
- 2 Check to make sure /u is the current working directory.
- 3 /u is the current working directory.
- 4 Create a new directory for *user1* setting the permission bits to 700. See "Controlling access to files and directories" on page 83 for information about permission bit settings.
- 5 List the contents of the /u directory.
- 6 The user1 directory entry.

The user file system that was previously created can now be mounted at /u/user1. The mount can be performed by:

- Using the /usr/sbin/mount REXX exec from the shell
- Using the TSO/E MOUNT command
- Using the ISHELL File_systems pull-down
- Adding an entry to the BPXPRMxx member in SYS1.PARMLIB so that it is remounted when the system reIPLs.

Figure 13 on page 135 shows an example of the commands required, including issuing the mount command from the shell. Type OMVS from ISPF option 6 to enter the shell and execute the highlighted commands to mount the file system OMVS.USER1.

```
# /usr/sbin/mount /u/user1 omvs.user1  1
OMVS.USER1 is now mounted at
/u/user1
# df -P                                2
Filesystem     512-blocks        Used  Available  Capacity Mounted
OMVS.USER1          12960          40      12920        1% /u/user1
OMVS.ROOT           82800       79608       3192       97% /
# chown user1:grpoe /u/user1           3
# ls -l /u/user1                       4
total 16
drwx------   2 USER1   GRPOE        0 Nov  7 09:09 user1
#

 ===>

ESC=¢ 1=Help    2=SubCmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Figure 13. Mounting the new file system

- 1 Issue the mount command to mount the file system, OMVS.USER1, on mount point /u/user1.
- 2 Run the display free space command to display the mounted file systems.
- 3 In order for USER1 to use this new file system, you must issue the chown command to change the ownership and to change the group to the user's default group. Issue this command to set the owner and group fields of this mount point directory for the USER1 ID. You only need to issue the chown command once because the values will be saved in the new file system and will be reused even when the file system is remounted later.
- 4 Issue a list command to display the new directory for USER1.

To make the mounting of the OMVS.USERS and OMVS.USER1 file systems permanent, add an entry in the BPXPRMxx member in SYS1.PARMLIB. These two mount statements must follow the ROOT statement for the root file system.

```
MOUNT    FILESYSTEM('OMVS.USERS')
         TYPE(ZFS)
         MOUNTPOINT('/u')
         MODE(RDWR)

 MOUNT   FILESYSTEM('OMVS.USER1')
         TYPE(ZFS)
         MOUNTPOINT('/u/user1')
         MODE(RDWR)
```

Using file locks

Programs using files can specify locks. Locks are used to lock byte ranges in files and are intended for use by cooperating application programs. For information about how systems participating in a shared file system handle file locks, see "Locking files in the sysplex" on page 188.

Programs can use file locks via fcntl().

- For more information about the fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors callable service, see *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.
- *z/OS XL C/C++ Runtime Library Reference* has details about the fcntl() function.

Locks are advisory. (Advisory locking is used in UNIX systems.) Consequently, more than one program can update a file at the same time. Keep advisory locking in mind during problem determination.

Creating special files

There are several types of special files. Pipes and FIFO special files are created by programs and users; character special files are typically created by the system programmer.

Restriction: You cannot share character special files in read/write mode among systems participating in a shared file system in a sysplex.

Pipes and FIFO special files are created by programs and users; character special files are typically created by the system programmer.

Character special files

A character special file is a file that provides access to an input/output device. Examples of character special files are: a terminal file, a NULL file, a file descriptor file, or a system console file. Each character special file has a device major number, which identifies the device type, and a device minor number, which identifies a specific device of a given device type. Character special files are customarily defined in /dev; these files are defined with the **mknod** command. You must have UID(0) to create a character special file. The best way to obtain UID(0) is to be defined to BPX.SUPERUSER FACILITY class. Then issue the **su** command to switch to UID(0) before issuing the **mknod** command.

You cannot share character special files in read/write mode among systems participating in a shared file system in a sysplex.

Pipe special files

Pipe special files are a way to communicate in first-in-first-out (FIFO) order from one or more processes to one or more processes. Pipes are treated as though they were files.

The following figure shows how a pipe works.

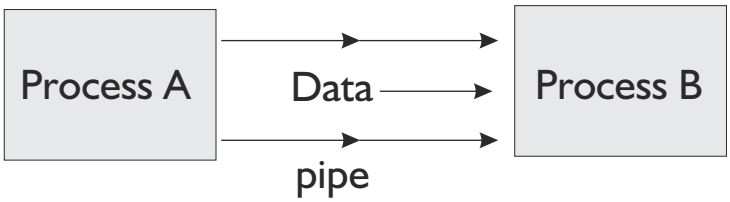


Figure 14. How a pipe works

A pipe sends data from one process to another or back to itself. By forking processes, a pipe can be shared by a number of processes; for example, written to by three processes and read by seven.

A program creates a pipe with a pipe() function. The pipe vanishes when the last process closes it. A pipe does not have a name in the file system; a pipe is also called an *unnamed pipe*.

FIFO special files

A FIFO special file sends data from one process to another so that the receiving process reads the data first-in-first-out (FIFO). A FIFO special file is also called a *named pipe,* or a *FIFO*. A FIFO special file can also be shared by a number of processes that were not created by forks. A FIFO special file can be written into and read by the same process using multiple threads.

FIFO special files can be shared between systems that use shared file systems. For more information about shared file systems, see Chapter 7, “Sharing file systems in a sysplex,” on page 155.

A program creates a FIFO special file with a **mkfifo** command or a mkfifo() function. The name is maintained in the file system until the named pipe is deleted by an **rm** command or an unlink() function.

UNIX domain socket address files

UNIX domain socket address files represent socket addresses in the UNIX domain. These files cannot be shared in read/write mode among systems participating in a shared file system in a sysplex.

To prepare for using AF_UNIX (local) sockets, the AF_UNIX physical file system (PFS) creates a socket address file in the file hierarchy during the bind() function call. The files are defined as specified by the program that calls bind() and are typically in the user's home directory, the root directory, or in `/tmp`. Because they are part of the file system, be careful not to delete any of these socket address files by accident. Also, do not unmount (manually or via automount) the file system that contains these socket files with active binds. If you do delete the socket files or unmount the containing file systems, then programs will not be able to connect to or send datagrams to the program that created the file.

Pseudoterminal files

Pseudoterminals (pseudo-TTYs) are used by users and applications to gain access to the shell. A pseudo-TTY is a pair of character special files: a *manager file*, and a corresponding *subsidiary file*.

- The manager file is used by a networking application such as OMVS or rlogin.
- The corresponding subsidiary file is used by the shell or the user's process to read and write terminal data.

The convention for the names of the pseudo-TTY pair is as follows:

- `/dev/ptyp`*NNNN* for the manager (major 1)
- `/dev/ttyp`*NNNN* for the subsidiary (major 2)

The *NNNN* is between 0000 and one less than the MAXPTYS value in the BPXPRMxx member.

When a user enters the TSO/E OMVS command or logs in using rlogin or telnet to initialize a shell, the system selects an available pair of these files. The pair represents the connection. The maximum number of pairs is 10000. You can specify an appropriate number of pairs in the MAXPTYS parameter; see “MAXPTYS” on page 28.

The default controlling terminal can be accessed through the `/dev/tty` special file (major 3). This file is defined the first time that the system is IPLed.

Pseudo-TTY files are dynamically created by the system when they are first referenced.

Null file

The null file, `/dev/null`, (major 4, minor 0) is analogous to an MVS DUMMY data set. Data written to this file is discarded. The standard null file, named `/dev/null`, is created the first time the system is IPLed, or when referenced, if it does not exist already.

Zero file

The zero file, `/dev/zero` (major 4, minor 1), is similar to `/dev/null` in that data written to this file is discarded, but when the file is read from, it provides an inexhaustible supply of binary zeros. The standard zero file, named `/dev/zero`, is created the first time the system is IPLed or when referenced, if it does not exist already.

Random number files

The random number files, `/dev/random` and `/dev/urandom` (major 4, minor 2) provide cryptographically- secure random output that was generated from the available cryptographic hardware. The foundation of this random number generation is a time-variant input with a very low probability of recycling.

Requirement: In order to use these device files, Integrated Cryptographic Service Facility (ICSF) must be started and the cryptographic hardware is required, depending on the model of the server. ICSF is not required if you are using an IBM z14® server or later. For more information about the requirements, see the usage notes in Random Number Generate in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

The hardware is designed to produce 8-byte random numbers but any amount of data might be read. Reads will fail if ICSF or the hardware is not available or if any addresses passed are invalid. Reads will not block. Data that is written to these devices will be ignored without being referenced.

These files are created whenever the system is started or when referenced if they do not exist. The default permissions are 666, RW-RW-RW-. You can change these permissions with **chmod** or by explicitly defining the devices with **mknod**.

Rules: Neither rule applies if you are using an IBM z14 server or later.

- To read from these devices, the user must be authorized to use ICSF, or ICSF must have been started with the CHECKAUTH(NO) option.
- For specific authority, if the CSFRNG resource in the CSFSERV class is protected, then the user must be permitted to the CSFRNG profile.

File descriptor files

A file descriptor file, `/dev/fd`*n* or `/dev/fd/`*n* (major 5, minor n) is used to refer to the same file as a previously opened file, as indicated by file descriptor *n*. If file *n* is a regular file or a character special file, the open for `/dev/fd`*n* or `/def/fd/`*n* is done as a real open of the file with file descriptor *n*. Otherwise, the dup protocol is used for that open.

When naming file descriptor file, the *n* in `/dev/fd`*n* or `/def/fd/` is the same as the minor number. The minor number determines which file descriptor number to duplicate. For example, opening `/dev/fd1` creates a file descriptor that is a duplicate of file descriptor 1. This might be useful for a program that expects a file name for output, but you might want it to write its output to stdout instead.

The `/dev/fd`*n* files are used by c89 to avoid the name-length limitations that are imposed by the DD statement PATH parameter.

Use of c89 assumes that you follow the naming conventions for file descriptor files.

File descriptor files are created dynamically as needed by the system when they are first referenced.

UNIX domain socket name special file

A path name specifies the socket address for a UNIX domain socket. The path name is assigned by the application programmer. There is no convention for the name. The operating system creates the file (major 6).

System console files

The following are system console files:

- `/dev/console` (major 9, minor 0). Data written to the `/dev/console` file is sent to the console and is displayed in message BPXF024I by means of a write-to-operator function. This message also contains the user ID of the process that wrote to the console. It is automatically created the first time the system is IPLed and is created with minor number 0.
- `/dev/operlog` (major 9, minor 1). This device file is intended for the syslog daemon. Data written to the `/dev/operlog` file is sent directly to the sysplex message log, OPERLOG, which must be active, and is displayed in message BPXF060I. For each write operation to `/dev/operlog`, the first character is removed from the message and used as a message indicator code with the following values:

  '00'x
  The message originated on a remote system.

  '80'x
  The message originated on the local system.

  Messages written to `/dev/operlog` that are not properly formatted results in an error return code with EINVAL as the errno.

  Using `/dev/operlog` is a quick way of logging messages.

Handling file system failures

If the file system fails, the operator must take several steps to restore it. See “File system failure” on page 280 for this information.

Restoring the root file system

If the physical file system owning the root fails, all work in progress when a failure occurs is lost, and it must be restarted from the beginning.

CAUTION: Unmounting and remounting a root file system is disruptive to the system. Any work in progress must be undubbed and redubbed.

The person who restores a failed root file system or an unmounted root file system must be a superuser who is defined with a home directory of / (root).

Recovering from file system problems with the root

If the root file system becomes corrupted, you can restore to the last known good copy and IPL the system, or you can avoid doing an IPL by following the steps described in this topic. This procedure, while it does not require an IPL, is disruptive to all UNIX processes. For example, any work that depends on TCP/IP will be affected.

Steps for recovering from file system problems with the root

Before you begin: You need to have a terminal that does not depend on TCP/IP, and the user ID doing the unmounts must be defined as UID(0) or have appropriate privileges under UNIXPRIV.

Perform the following steps to recover from file system problems with the root.

1. List the applications that are running. Issue:

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

   _____

2. Bring down all the processes listed, except for BPXOINIT.

   _____

3. Make sure that all the processes except for BPXOINIT have been shut down. Issue:

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

   The display will look like the following:

```
BPXO040I 07.31.14 DISPLAY OMVS 017
OMVS     000E ACTIVE          OMVS=(65)
USER       JOBNAME  ASID        PID       PPID STATE      START     CT_SECS
IBMUSER  BPXOINIT   0013          1          0  MR---B     07.21.27     .034
LATCHWAITPID=         0 CMD=BPXPINPR
SERVER=Init Process           AF=    0 MF=00000 TYPE=FILE
```

   _____

4. Identify the file systems that are mounted.

```
D OMVS,F
```

   You will see a display that shows the mounted file systems.

   _____

5. Unmount the file systems by using the TSO/E ISHELL command.

   The root file system must be unmounted last and you must use the IMMEDIATE option when unmounting the root file system. After the root has been unmounted, the mount table should show SYSROOT.

   _____

6. Check the display again.

```
D OMVS,F
```

   You should see a display similar to the following:

```
BPXO044I 10.38.16 DISPLAY OMVS 054
OMVS     000E ACTIVE          OMVS=(65)
TYPENAME   DEVICE ----------STATUS----------- MODE QJOBNAME  QPID
BPXFTCLN    0 ACTIVE                       RDWR
  NAME=SYSROOT
  PATH=/
```

   _____

7. Follow your recovery actions for the root file system.

   _____

8. Mount the root. For example, from TSO READY or ISPF Option 6, issue:

```
MOUNT FILESYSTEM('your root dsname') TYPE(ZFS) MOUNTPOINT('/')
```

   The root is mounted at / in read/write mode. If your root is read-only, add MODE(READ) to the MOUNT FILESYSTEM command.

   _____

9. Mount the individual file systems at their respective mount points by using the TSO/E ISHELL command.

_____

10. After the file systems are mounted (check by issuing D OMVS,F again), have a superuser (UID=0) enter the shell and issue `/etc/rc` from the prompt to run the shell initialization script.

    _____

11. Follow your procedures to restart other applications (such as TCP/IP, NFS, FTP, and WebServer) and confirm that all functions are working.

    _____

You know you are done when you have rebuilt the z/OS UNIX environment and all functions are working.

Installing service

Some customers install service in response to a particular problem that they experience. Others install preventive service to prevent problems from occurring.

To install service, create a copy of the system that you are migrating from onto another set of volumes. Sometimes the system that you are migrating from is the active production-level system, called the *driving system*. This new copy is called the *target system*. The DDDEF entries or the DD statements in the cataloged procedure that is used when applying service are updated to point to the libraries on the target system. When service is applied, updates are made to the target libraries.

After service is installed, the new target libraries are tested before being put into production as the new driving system.

As you prepare to install service into the file system, keep the following facts in mind:

- There is only one file hierarchy active at any given time. You might have multiple file systems on your system. But z/OS UNIX does not recognize them unless they are mounted at a directory (mount point) within the file hierarchy.
- If you install service directly on the production file system, you will copy new load modules over existing ones, which might cause potential tracking and system-level problems. Therefore, you should create a copy of the production file system before installing service.

  In most cases, you will copy the root file system. However, you can use this same concept to duplicate other production file systems that are mounted in the file hierarchy or in individual directories. For help copying the root file system, see “Copying the file system” on page 129.

  This new copy must be mounted at a directory (mount point) within the active file hierarchy. The directories in the newly mounted file system will be the target libraries when installing service.
- The distribution libraries for elements installing into the file system are still partitioned data sets.

Steps for installing service into the z/OS UNIX file system

Follow these steps each time you install service into the z/OS UNIX file system.

About this task

The new file system is called the *service file system*. The first two steps are described further in Figure 15 on page 142.

Procedure

1. Create a clone of the system that you are migrating from. Creating a clone includes copying all necessary partitioned data sets and file systems. Use utilities such as IEBCOPY or DFSMSdss to copy the partitioned data sets.
2. With a superuser ID, mount the service file system at a mount point in the active file hierarchy. In the following example, the name `/service` is used, but you can use a different name.

To mount the service file system, first create a directory for the `/service` directory by following these steps:

a) Issue the following TSO MKDIR command to create a directory called `/service`.

```
MKDIR '/service'
```

b) Issue the TSO/E MOUNT command to mount the service file system to the root file system.

The `/service` directory now has permission bits of 755, which prevents unauthorized users from corrupting the service file system.

3. Change the DDDEF entries or DD statements that are used by the SMP/E cataloged procedure to point to the new target directories.
   For example, the DDDEF entries must now point to `/service/bin/IBM` instead of to `/bin/IBM`.
   With SMP/E, you can perform the ZONEEDIT function for all directories. You no longer need to manually change individual DDDEF entries for all directories.
4. Change the VOLSER information of the DDDEF entries or DD statements for the partitioned data sets on the target volumes.
5. Install the service.
6. Test the new target libraries. Keep the SYSRES of the target system and the file system of the target system synchronized because service might affect both files in the file system and members of the partitioned data set. Make both the target system SYSRES and the target file system available at the same time.
7. Move the target libraries into production. To replace the original file system with the service file system, use one of the following methods:
   a) Use the DFSMSdss DUMP and RESTORE commands (or another appropriate utility) to copy the service file system to the original file system.
   b) Unmount the original file system. Then, unmount the service file system from `/service` and mount it on the mount point of the original file system. A reIPL is required.

Results

You have installed service into the z/OS UNIX file system.

Example of installing service

Figure 15 on page 142 shows a target system that was created by making a copy of the driving system to target volumes and by making a copy of the root file system. The root file system is copied into another file system and is mounted within the file hierarchy. In this example, it is mounted to the `/service` directory.

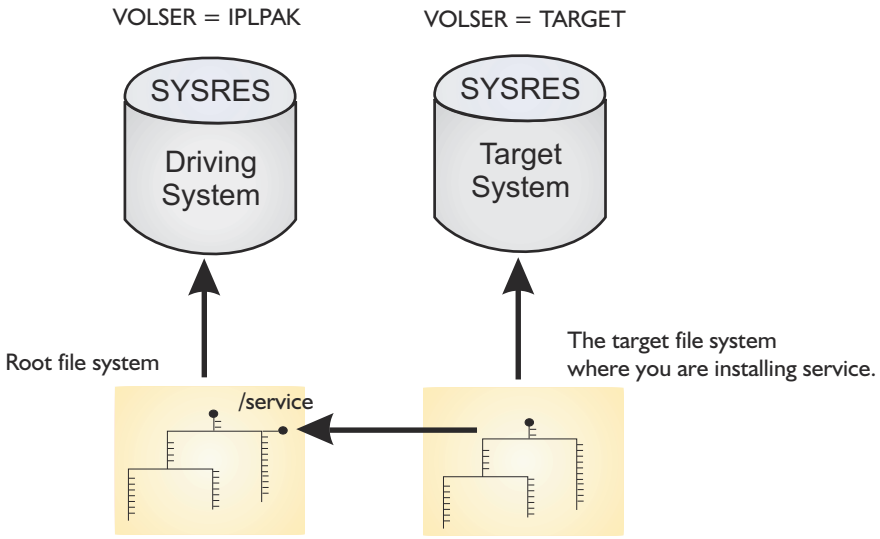


Figure 15. Preparation for installing service

The DDDEFs or the DD statements used by the cataloged procedure when applying service must point to the target system.

The following example shows sample DDDEFs pointing to new target libraries:

```
LPALIB   DD  DSN=SYS1.LPALIB,VOLSER=TARGET,
             UNIT=3390,DISP=SHR

SFSUMBIN  DD  PATH='/service/bin/IBM'
```

Transporting the file system from the driving system to the target system

It is possible to install products and service on one system and then transport this system image to the rest of their enterprise. Using the DFSMSdss DUMP and RESTORE commands (or another appropriate utility), you can dump individual product libraries or full volumes, transport them to other systems, and restore them.

However, because the individual file systems that make up the active file hierarchy might be on SMS-managed volumes, there are some special considerations for making a transportable copy.

- You can dump each file system to be transported into individual sequential data sets by using the DFSMSdss dump utility. These sequential data sets contain all the necessary information about the files and can also exist with other product libraries that need to be transported.
- After the system image has been transported to the target system, you can restore individual product libraries or full volumes by using the DFSMSdss RESTORE command.
- After the data sets have been unloaded on the target system, you can use the DFSMSdss RESTORE command to restore the sequential data sets into individual file systems. These file systems will make up the active file hierarchy on the target system.

With this process, you can duplicate system images across the enterprise.

For information about the DFSMSdss utilities, see *z/OS DFSMSdss Storage Administration*.

Making changes to /etc and /var

Because `/etc` and `/var` are symbolic links, and not directories, this difference affects how you install service and make updates while you are doing a system replace. Follow the procedure in "Steps for making changes to /etc and /var" on page 144 if the installation process must create one or more directories, or if you would like to copy files from one `/etc` or `/var` file system to another`/etc` or `/var` file system.

The `/etc` file system is the location for your own customization data for products. You set up the `/etc` files and you maintain their content. The `/var` file system is the location for IBM product information that is created, used, and maintained by IBM products. IBM products might create directories under `/etc` or `/var` during installation, but IBM does not create files under `/etc` or `/var` during SMP/E installation. Because IBM products do not create files into `/etc` or `/var`, there is no possibility that SMP/E installation of an IBM product or service will overlay your own files within `/etc` or `/var`.

For information about how to migrate the `/etc` and `/var` file systems during a system replace, see *z/OS Upgrade Workflow*.

If you are mounting a maintenance file system that is named MAINT.ETC at a symbolic link (`/service/etc`) that points to the `/SYSTEM/etc` mount point where the active file system (PROD.ETC) is mounted, you must convert the symbolic link to a directory first.

Steps for making changes to /etc and /var

Follow these steps each time you make changes to /etc or /var.

About this task

Because /etc and /var are symbolic links, not directories, they cannot receive product or service code. If you are installing service or products that must write to /etc or /var, you need to change /etc or /var to a directory, install the code, and then change /etc or /var back to a symbolic link. The sample JCL for those tasks is in SYS1.SAMPLIB; they are:

While you can mount on a symbolic link that points to a directory, you must make sure that the mount point that your symbolic link is pointing to is not still mounted on by the original /etc file system.

Procedure

1. Mount a clone of the system that you are installing into the /service mount point.
2. Run the sample job BPXISETD to convert the /service/etc and /service/var symbolic links to directories. Pass the /service parameter to the REXX exec.

   Optionally, you can use BPXISJCL to submit the job in the background.
3. Mount the clone of the serviceETC.ZFS of the system you need to service at /service/etc. For /var directory updates, mount the clone of serviceVAR.ZFS of the system you need to service at /service/var.
4. Install the service, which might include running REXX execs that create directories under /service/etc or /service/var. At this point, you can copy /etc files from an existing file system's /etc to /service/etc, if you wanted to bring forward any configuration files, and change them properly.
5. Unmount the /etc after everything is installed from /service/etc. Unmount the /var after everything is installed from /service/var.
6. Run the sample job BPXISETS to convert the /etc file system back to the /etc symbolic link.

Results

When you are done, you have installed product or service code into the /etc directory.

Installing products into the file system

When you install other products into the file system, you will have to create new directories where the files that are associated with the new product will be installed. You might also need to create a new file system for the new product and mount it to a new directory.

To help you decide how many file systems you need, see Placing data sets on specific volumes in *z/OS Planning for Installation*.

Steps for installing products into the z/OS UNIX file system

Follow this procedure if you are installing products into the /etc file system on a system.

About this task

Because /etc is a symbolic link, it cannot receive product or service code. You must run jobs that change /etc to a directory before you can install the code. Then, you need to change /etc back to a symbolic link.

Follow these steps each time you install product or service code into the /etc directory. The sample jobs can be found in SYS1.SAMPLIB.

Procedure

1. Mount a clone of the root file system at the `/service` mount point.
2. Run the sample job BPXISETD to convert the `/etc` symbolic link to the `/etc` directory. Pass the `/etc` directory as a parameter to the REXX exec. You can also use the BPXISJCL sample job to submit the job in the background.
3. Mount the `/etc` file system.
4. Install the products or service.
5. Unmount the `/etc` file system after you install all the products or service.
6. Run the sample job BPXISETS to convert the `/etc` file system back to the `/etc` symbolic link.

Results

When you are done, you have installed product or service code into the `/etc` directory.

Chapter 6. Using the automount facility

The automount facility automatically mounts file systems at the time they are accessed. It manages the creation of the mount point and the mount of the user file system for you. Whenever someone accesses a directory that is managed by the automount facility, the mount is issued automatically.

Using the automount facility provides the following advantages:

- It is easier to manage file systems. You do not need to mount most file systems at initialization. You also do not need to request that operators perform mounts for other file systems. It is easier to add new users because you can keep your parmlib specification stable. You can establish a simple automount policy to manage user home directories.
- Resources are not consumed until they are requested. A file system that is managed by the automount facility remains unmounted until its mount point is accessed.
- You can reclaim system resources used by a mount if that file system has not been used for a period of time. You can specify how long the file system should remain mounted after its last use.

You can also use the automount facility for other file systems such as NFS.

If you are using system-specific security labels, do not use the automount facility.

Automounting zFS file systems

You can use a single automount policy to manage zFS file systems in the same managed directory. An *automount policy* specifies the file systems that are to be mounted by the automount facility. For more information about the automount facility, see Chapter 6, “Using the automount facility,” on page 147.

You can use the IBM Health Check USS_AUTOMOUNT_DELAY to evaluate the delay times of the configuration. For more information, see USS_AUTOMOUNT_DELAY in *IBM Health Checker for z/OS User's Guide*.

Automounting NFS file systems

You can use the automount facility for Network File System (NFS). For information about the parameter requirements for NFS client mounts, see Accessing z/OS UNIX file systems and z/OS conventional MVS files in *z/OS Network File System Guide and Reference*.

How does the automount facility work?

You create an automount policy that specifies directories containing only mount points, which is the recommended method of managing file systems. An automount policy specifies the file systems that are to be mounted by the automount facility. As each mount point is accessed, an appropriate file system is mounted. The mount point directories are created as they are required. If the file system is no longer used, the mount point directories are deleted.

Think of the automount facility as an administrator that has total control over a directory. When a name is accessed in this directory, the automount facility checks the automount policy for the file system that is supposed to be associated with that name. Then, it performs a **mkdir** followed by a mount and moves out of the way. Now the root directory of that newly mounted file system can be accessed as that name.

For example, suppose that you had created the USER1 directory with the **mkdir** command. If you had set up the automount facility and put the automount policy in place, you would not have needed to do that. The USER1 directory would have been dynamically created and the OMVS.USER1 data set automatically mounted at the `/u/user1` mount point.

Later, if the `/u/user1` file system was not accessed based on certain criteria in your automount policy, the OMVS.USER1 data set is automatically unmounted and the USER1 directory removed.

The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

You can use a prefilter by updating the automount master file to include the name of the filter utility. For more information, see the automount: Configure the automount facility description in *z/OS UNIX System Services Command Reference*.

The automount file system is mounted with an automove attribute of either AUTOMOVE or UNMOUNT. The automove attribute is set to UNMOUNT only when its parent file system has its automove attribute set to UNMOUNT.

If the automount policy is loaded, you will get a return code of 0. A nonzero return code indicates that the policy was not loaded.

Setting up the automount facility

You need to customize the configuration files before you can use the automount facility.

- The `/etc/auto.master` file must contain the directory or directories that the automount facility will monitor.
- The MapName file must have entries for the mapping between the subdirectory of a directory managed by the automount facility and the mount parameters. For an example of a MapName file, see "MapName" on page 148.

You can also refer to automount: Configure the automount facility in *z/OS UNIX System Services Command Reference* for more information about both files.

File system name templates that use symbol symbolics cannot be more than 44 characters long. Symbolics used for the automount facility (&SYSNAME., `<asis_name>`, `<us_name>`) are resolved within automount as part of checking the length of the file system name template.

/etc/auto.master

The `/etc/auto.master` file contains the directory or directories that the automount facility will monitor. It also contains an associated MapName file that contains the mount parameters.

The following example shows a `/etc/auto.master` file. It specifies that the automount facility is to manage the `/u` directory. If someone using kernel services tries to access a directory in the `/u` directory, the automount facility automatically mounts the data set based on the MapName policy in Figure 17 on page 149.

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

Figure 16. Example of a /etc/auto.master file

The name of the map file can be specified as a data set name. The data set name must be specified as a fully qualified name and can be uppercase or lowercase. Single quotation marks are not needed. For example:

```
/u //sys1.parmlib(amtmapu)
```

MapName

The MapName file contains the mapping between a subdirectory of a directory managed by the automount facility and the mount parameters. It can contain both specific entries and a generic entry. When the automount facility tries to resolve a lookup request, it attempts to find a specific entry. If a specific entry does not exist for the name being looked up, it will then attempt to use the generic entry.

Tip: The MapName file can contain only one generic entry, and it has to be the first entry in the MapName file. When using generic entries, you should have a consistent naming criterion. The file system in Figure 17 on page 149 has a high-level qualifier of OMVS, and the lower level qualifier is equal to the user ID.

Figure 17 on page 149 shows an example of a MapName file. It contains the mount parameters for the user directories.

```
name               *
type               ZFS
filesystem         OMVS.&SYSNAME..<uc_name>
mode               rdwr
duration           nolimit
delay              10
setuid             no | yes
```

Figure 17. Example of a generic entry in a MapName file, /etc/u/map

In the example, &SYSNAME. represents the system name while <uc_name> specifies that the name being looked up is to be represented in uppercase. The automount facility creates a directory containing that name and uses it as a mount point for the file system to be mounted. You can use <uc_name> to replace any level qualifier. For example, if the name of the directory that is being looked up is USER1, the automount facility will resolve the name in the following ways:

```
OMVS. <uc_name> = OMVS.USER1
OMVS. <uc_name>.ZFS = OMVS.USER1.ZFS
```

For a complete list of supported keywords, see automount: Configure the automount facility in *z/OS UNIX System Services Command Reference*.

Security considerations for the automount policy

In the MapName file, the setuid keyword specifies whether to support or ignore the setuid or setgid mode bits on executable files loaded from the file system. The default is yes.

For security reasons, consider specifying `setuid no` . If you do, then the setuid and setgid flags in the permission bits are ignored, as well as the program control extended attribute (+p) and the APF-authorized extended attribute (+a). Consider the following:

- z/OS UNIX files and directories are contained in MVS data sets.
- z/OS UNIX users using these files and directory do not need access to these MVS data sets. Only the kernel and your storage administrators need access to the data sets.
- If you give the users direct access to the MVS data sets by giving them UPDATE access in a RACF profile protecting the data sets, or by naming the data sets with the user ID as the HLQ, and you do not specify `setuid no` when mounting, you have a security exposure.

Using map files from other systems

If you have been using a map file on another system and want to use it on z/OS UNIX, you will have to use a conversion utility to reformat it. If you want to use a map file with syntax different from that supported by z/OS UNIX, such as a map file from another type of system, you can implement a conversion utility that converts the file to syntax supported by z/OS UNIX. Then you can direct the automount utility to run it and use the output of your conversion utility as the map file. For more information, see the automount: Configure the automount facility in *z/OS UNIX System Services Command Reference*.

Steps for setting up the automount facility

Before you begin: You must have a superuser ID in order to activate the automount facility from the shell.

Perform the following steps to set up the automount facility.

1. Add the following statement to your BPXPRMxx parmlib member.

   ```
   FILESYSTYPE TYPE(AUTOMNT) ENTRYPOINT(BPXTAMD)
   ```

   ——
2. Either restart z/OS UNIX or issue SETOMVS RESET to activate the automount PFS.

_____

3. Customize the `/etc/auto.master` file. For more information about the `/etc/auto.master` file, see "/etc/auto.master" on page 148 and *z/OS UNIX System Services Command Reference*.

   Set the permission bits so that the file is protected from write by ordinary users like other system files. The files should be owned by UID(0) and have write permission only for owner, such as 644. If the group for the file is a properly restricted group, 664 would also be appropriate.

   _____

4. Customize the MapName file

   _____

5. Activate the automount facility from the shell. You can activate the automount facility in one of two ways as shown in Table 19 on page 150. Base your choice on your particular situation.

   Table 19. Ways of starting the automount facility. This table lists the ways of starting the automount facility (from the shell and during initialization)

If you want to start the automount facility	Then
From the shell **You must have a superuser ID.**	Issue: `/usr/sbin/automount` If the automount facility was started from the shell, do not submit any job that requires an automount-managed file system until automount initialization is complete.
During initialization	Add the following lines to the `/etc/rc` file: `# Start the automount facility` `/usr/sbin/automount` If the automount has been started from `/etc/rc`, do not submit jobs that require an automount-managed file system before you get the message BPXI004I, or they might fail due to allocation errors.

   When working with the automount facility:

   - If arguments are not used, the automount facility reads the `/etc/auto.master` and MapName files. If a master file name is specified, that file name is used instead of `/etc/auto.master`.
   - The `-s` option only checks the syntax of the configuration file. The automount policy is not activated.
   - Use the `-a` option if you want to append the new automount policy to the original policy instead of replacing it. It is mutually exclusive with the `-q` option.

   _____

When you are done, you have activated the automount facility.

What happens when you start the automount facility from the shell?

The following figure shows what happens when the automount facility is started from the shell. It shows how file systems are automatically mounted. To access the shell, type OMVS from ISPF option 6.

```
# df
Mounted on     Filesystem                Avail/Total    Files      Status
/              (OMVS.ROOT)               1432/89280     0          Available
# /usr/sbin/automount                    1
FOMF0107I Processing file /etc/u.map
FOMF0108I Managing directory /u            2
# df                                3
Mounted on     Filesystem                Avail/Total   Files       Status
/u             (*AMD/u)                  0/8           0           Available
/              (OMVS.ROOT)               1432/89280    0           Available
# cd /u/user1                              4
# cd /u/slekka/testdir                     4
# cd /u/rpetri                             4
# df                                       5
Mounted on     Filesystem                Avail/Total    Files      Status
/u             (*AMD/u)                  0/8            0          Available
/u/rpetri      (OMVS.RPETRI)             4256/4320      0          Available
/u/slekka      (OMVS.SLEKKA)             4232/4320      0          Available
/u/user1       (OMVS.USER1)              4232/4320      0          Available
/              (OMVS.ROOT)               1432/89280     0          Available
# ls -l /u                                 6
Total 496
drwxr-xr-x   2 RPETRI   OMVSGRP       0 Nov  2 09:59 rpetri
drwxr-xr-x   2 SLEKKA   OMVSGRP       0 Nov  1 09:47 slekka
drwx------   2 ADMIN    OMVSGRP       0 Nov  7 09:07 user1
# chown user1 /u/user1                     7
# ls -l /u                                 8
Total 496
drwxr-xr-x   2 RPETRI   OMVSGRP       0 Nov  2 09:59 rpetri
drwxr-xr-x   2 SLEKKA   OMVSGRP       0 Nov  1 09:47 slekka
drwx------   2 USER1    OMVSGRP       0 Nov  7 09:07 user1
#

 ===>

ESC=¢ 1=Help    2=SubCmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FedRetr 12=Retrieve
```

Figure 18. Follow-up steps when using the automount facility

- 1 The `automount` command is being issued from a superuser ID to start the automount facility from the shell.
- 2 The automount facility scans the `/etc/auto.master` file first to see what MapName file or files should be read. Here, the `/u` directory is being managed.

  Calling the `automount` command twice by mistake does not cause problems regardless of whether a file system is already mounted. The automount facility reads the `/etc/auto.master` file and associated MapName file or files again and then picks up any changes.
- 3 The display free space command (`df`) is issued. It shows that the automount facility has been started and is managing the `/u` directory. Notice the `(*AMD/u)`.
- 4 Change directory (`cd`) commands are issued to access directories in the three file systems that are to be mounted from the `/u` directory. In this case, the directories USER1, RPETRI, and SLEKKA are used to resolve the <uc_name> symbol in the `/etc/u.map` file. The RPETRI, SLEKKA, and USER1 directory names are translated to uppercase and substituted to build the data set names OMVS.RPETRI, OMVS.SLEKKA, and OMVS.USER1, respectively. The RPETRI, SLEKKA, and USER1 directories do not physically exist in any file system but will be created as pseudo mount points by the automount facility on which the zFS data sets OMVS.RPETRI, OMVS.SLEKKA, and OMVS.USER1 are mounted.
- 5 Output from another `df` command shows that (*AMD/u) is managing the `/u` directory. It also shows that the OMVS.RPETRI, OMVS.SLEKKA, and OMVS.USER1 data sets are now mounted at pseudo mount points `/u/rpetri`, `/u/slekka`, and `/u/user1`, respectively.

  When automount is actively managing a particular mount point (in this case `/u`) you cannot add a file to this directory (`/u`) or create a new subdirectory off the `/u` directory using the `mkdir` command. If you try, you will see an allocation or catalog error.
- 6 The `ls -l /u` command is issued against the `/u` directory and the directory attributes are displayed.
- 7 The `chown` is issued to change the ownership of `/u/user1` directory from ADMIN to USER1.

- 8 The `ls -l /u` command is issued again to show that the owner field of the `/u/user1` directory is now set to USER1.

Figure 18 on page 151 shows how <uc_name> works with the `/etc/auto.master` and `/etc/u.map` files from Figure 16 on page 148 and Figure 17 on page 149. The OMVS.RPETRI, OMVS.SLEKKA, and OMVS.USER1 data sets have already been allocated. The low-level qualifier of the data sets is the user ID, which is also the directory mount point that the automount will dynamically allocate. With the automount facility, if a user tries to access any directory in their file system, the data set is automatically mounted under the `/u` directory.

When working with the automount facility:

- You can use specific entries for directory names when the parameters you want to use differ from the generic entry. Any parameters that are not specified are inherited from the generic entry. A specific entry defines a directory name that is called `wjs` in the name parameter of the MapName file rather than an `*` as shown in Figure 19 on page 153.

  In this example, the duration for generic mounts is set to unmount idle file systems after 60 minutes. But in this specific mount entry, idle file systems will stay mounted indefinitely. Also, a specific file system is specified because the file system name does not conform to the format in the generic entry. All other attributes are inherited from the generic entry.
- To display the current automount policy, issue

```
/usr/sbin/automount -q
```

- You can specify the allocation parameters when using `automount` to allocate a data set. The specifications for the automount map file have keywords that you can use to specify allocation keywords.

Note:

1. When a new file system of the type ZFS is created and allocated to a new user, the owner UID and GID are based on that user. The setting of the permission bits is 700. By default, the automount process uses the UID and GID of the user ID that owns the process. If the `euid` keyword is specified for `allocany` or `allocuser`, the thread level UID and GID are used instead.
2. When a new file system of the type ZFS is created and allocated to a new user, the owner UID and GID are based on that user. The setting of the permission bits is 750. By default, the automount process uses the UID and GID of the user ID that owns the process. If the `euid` keyword is specified for `allocany` or `allocuser`, the thread level UID and GID are used instead.
3. If the `ucat` keyword is used with the allocation parameter, the high-level qualifier of the new zFS data set must be an alias in the master catalog, and that alias must point to a user catalog.

Naming specific directories

Given the `/etc/auto.master` and `/etc/u.map` files as shown in Figure 19 on page 153, whenever the directory `/u/wjs` is referred to by a command such as **cd** or **cp**, the automount facility mounts the OMVS.WJS.ZFS data set.

```
------ /etc/auto.master  ------

/u         /etc/u.map


 ------ /etc/u.map  ---------

name               *
type               ZFS
filesystem         OMVS.<uc_name>
mode               rdwr
duration           60
delay              10
/*
name               wjs
filesystem         OMVS.WJS.ZFS
duration           nolimit
```

Figure 19. Specific entry in a MapName file

For generic and specific entries:

- Do not use a / in front of the name of the directory to be mounted in a specific entry in a MapName file. For example, in `/etc/u.map`, the following is correct:

  ```
  name         wjs
  ```

  But the following is not correct:

  ```
  name         /wjs
  ```

- The directory name and the data set name qualifier for the data set that is replaced by the variable *<uc_name>* must be the same. Otherwise, you will receive such error messages as:

  ```
  DC129I No such file or directory
                 or
  EDC515I Dynamic allocation error
  ```

Changing which file systems are automounted

The automount facility is a physical file system (PFS) that is started with a FILESYSTYPE statement in the BPXPRMxx parmlib member. After the PFS is started, the automount facility manages the policy that you make active by using the `/usr/sbin/automount` command. You can change the automount policy at any time, although you cannot set it to null. To change it, update the automount configuration files (`/etc/auto.master` and MyMap). Then activate that configuration with the `/usr/sbin/automount` command.

Stopping the automount facility

You cannot stop the automount facility after it has been started. Instead, change the automount configuration files (`/etc/auto.master` and MyMap) so that the automount facility is set to manage a dummy directory. Then activate that configuration with the `/usr/sbin/automount` command. Or you can simply unmount any `*AMD/` file system.

Automounting in a shared file system environment

The most recent automount policy that was loaded prevails for all the systems participating in a shared file system.

Rule: You must keep the automount policy consistent across all the participating systems in the sysplex. The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

In order to keep the automount policy consistent across systems participating in a shared file system, ensure that each system has the same file contents for `/etc/auto.master` and `/etc/u.map`. Alternatively, a common repository for the automount policy can be created as follows:

1. Create a file called `/global/auto.master`. Its contents, for example, would be similar to the one shown in the following example. (This file would be equivalent to the `/etc/auto.master` file.)

   ```
   /u               /global/u.map
   ```

2. Create the `/global/u.map` file and place the automount policy in this file. (This file would be equivalent to the `/etc/u.map` file).
3. Then, on every system participating in a shared file system, create a symbolic link for `/etc/auto.master` to `/global/auto.master`. Also, delete each system-specific `/etc/u.map` file.

For more information about the `/global` directory, see Chapter 7, “Sharing file systems in a sysplex,” on page 155

The default delay time for automount is 10 minutes. Do not use a value less than 10. You can use the USS_AUTOMOUNT_DELAY check that is provided by IBM Health Checker for z/OS to verify the setting of the delay time. For more information, see USS_AUTOMOUNT_DELAY in *IBM Health Checker for z/OS User's Guide*.

Chapter 7. Sharing file systems in a sysplex

You will read about the shared file system capability in a multisystem sysplex environment. It is assumed that you already have completed the other setup activities for a sysplex environment. You will learn about the shared file system concept, the different file systems that exist in a sysplex, and how to establish that environment.

Although it is suggested that you exploit shared file system support when running in a sysplex environment, you are not required to do so. If you choose not to, you will continue to share file systems as you have before. To see how the file system structure has changed to support the shared file system environment, even when running on a single system, see "Illustrating file systems in single system and sysplex environments" on page 157.

z/OS Program Directory describes how IBM's integration test team implemented a shared file system.

Use the IBM Health Checker for z/OS to check the file system configuration, as described in Chapter 23, "IBM Health Checker for z/OS," on page 397.

What does shared file system mean?

By establishing the shared file system environment, sysplex users can access data throughout the file hierarchy from any system in the sysplex.

The best way to describe the benefit of this function is by comparing what was the file system sharing capability prior to the introduction of shared file system support 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 file system support, all file systems that are mounted by a system participating in a shared file system 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 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.

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 a shared file system.

There is also greater availability of data in case of system outage, and a 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 shared file system

You will read about the kinds of file systems and data sets that support the shared file system capability in the sysplex. Figure 20 on page 156 shows that, to the end users, the logical view of the hierarchical file system does not change. 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.

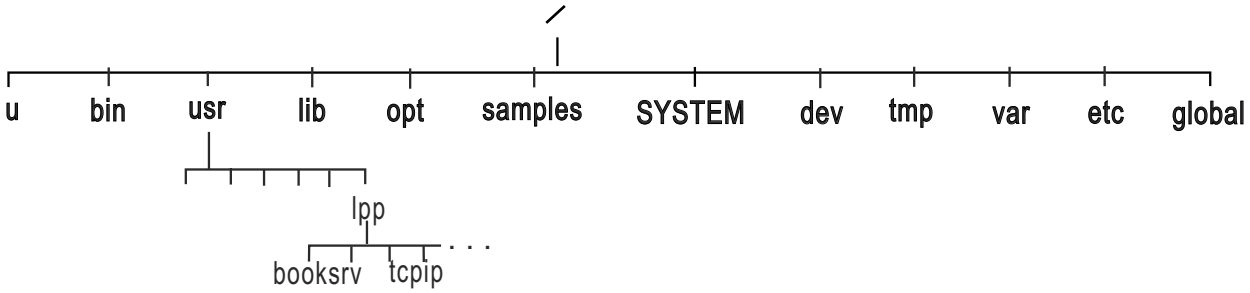


Figure 20. Logical view of a shared file system for the end user

This logical view applies to the end user only. However, system programmers need to know that the illustration of directories found in Figure 20 on page 156 does not reflect the physical view of file systems. For example, some of the directories are actually symbolic links.

Summary of various file systems in a shared environment

Table 20 on page 156 summarizes the file systems that are needed in a sysplex environment. As you study the illustrations of file system configurations, you can refer back to this table.

Table 20. Various file systems that exist in a sysplex. This table lists a summary of the various file systems in a shared environment

Name	Characteristics	Purpose	How created
Sysplex root	The sysplex root contains directories and symbolic links that allow redirection of directories. Only one sysplex root file system is allowed for all systems participating in a shared file system.	The sysplex root is used by the system to redirect addressing to other directories. It is very small and is mounted read-only.	For the zFS file system, the user runs the BPXISYZR job.

Table 20. Various file systems that exist in a sysplex. This table lists a summary of the various file systems in a shared environment *(continued)*

Name	Characteristics	Purpose	How created
System specific	The system-specific file system contains data that is specific to each system, including the `/dev`,`/tmp`, `/var`, and `/etc` directories for one system. There is one system-specific file system for each system that is participating in a shared file system. If you are using the TSO/E MOUNT command to set up system-specific file systems, specify the UNMOUNT parameter. Then, if data sets are replaced during a reIPL, the new data sets are mounted as the original file systems. Use the NORWSHARE mount parameter to mount zFS system-specific file systems.	The system-specific file system 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.	For the zFS file system, the user runs the BPXISYZS job on each participating system.
Version **In a sysplex, the *version file system* is the new name for the root file system.**	The version file system contains system code and binary files, including the `/bin`,`/usr`, `/lib`, `/opt`, and `/samples` directories. IBM delivers only one version file system; you might define more as you add new system levels and new maintenance levels.	The version file system has the same purpose as the root file system in the non-sysplex world. It must be mounted read-only. See "Mounting the version file system" on page 162 for a complete description of the version file system.	IBM supplies this file system in the ServerPac. CBPDO users create the file system by following steps that are defined in the Program Directory.

Illustrating file systems in single system and sysplex environments

The illustrations in this section show you how the file system structures have changed since the introduction of the shared file system support. These illustrations build upon IBM's long-standing suggestions 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 file system environment, that separation of system limit parameters from file system parameters is even more important. Each system will continue to have a system limits BPXPRxx member. As you will see in sections that follow, with shared file system support, you can have a file system BPXPRMxx member for each participating system or you can replace those individual file system BPXPRMxx members with a single file system BPXPRMxx member for all participating systems.

File systems in single system environments

Figure 21 on page 158 shows what BPXPRMxx file system parameters would look like in a single system environment, and Figure 22 on page 159 shows the corresponding single system image. SYSPLEX(NO) is specified (or the default taken), and the mount mode is read-only.

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

```
BPXPRMxx

FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

SYSPLEX(NO)

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


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE

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

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

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

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

Figure 21. BPXPRMxx parmlib member for a single system

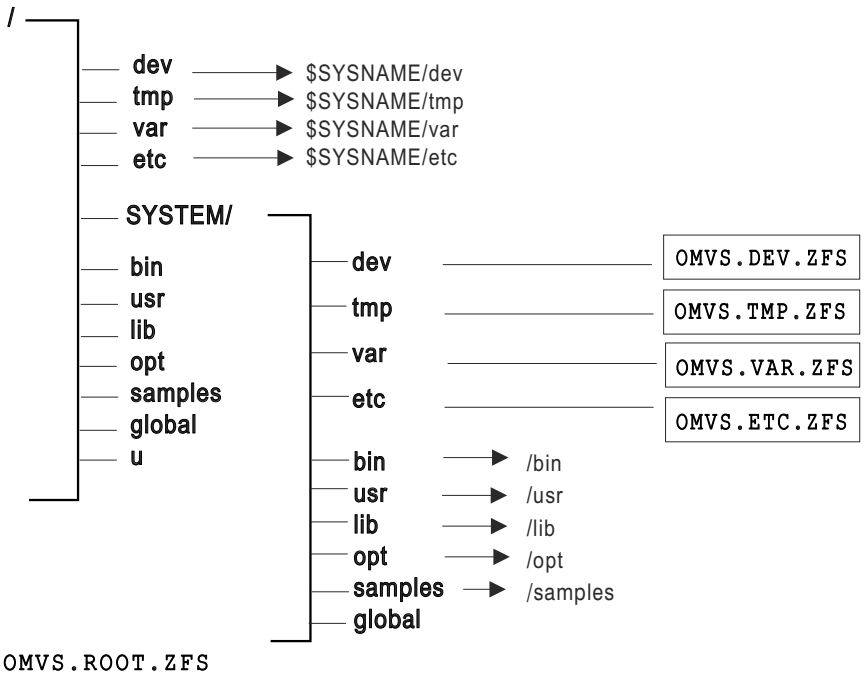


Figure 22. Illustration of a single system

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

In Figure 22 on page 159, 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.

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.

Establishing a shared file system in a sysplex

When setting up a shared file system in a sysplex, do not assume that with shared file systems, two systems can share a common file system for /etc, /tmp, /var, and /dev. This is not the case. Even with shared file systems, each system must have specific file systems for each of these mount points. The file systems are then mounted under the system-specific file system (see Figure 31 on page 176). With shared file system support, one system can access system-specific file systems on another system. (The existing security model remains the same.) For example, while logged onto SY2, you can gain read/write access to the /tmp for SY1 by specifying /SY1/tmp.

Be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in a shared file system. The easiest way to accomplish this is to create a single BPXPRMxx member that contains file system information for each system participating in a shared file system. 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 173.

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.

Tip: When using the TSO/E MOUNT command to set up system-specific file systems, specify the UNMOUNT parameter. Then, if data sets are replaced during a reIPL, the new data sets are mounted as the original file systems.

Creating the sysplex root file system

The sysplex root is a file system that is used as the sysplex-wide root. This file system must be initially mounted read/write and designated AUTOMOVE. (See "Customizing BPXPRMxx for a shared file system" on page 165 for a description of the AUTOMOVE parameter in BPXPRMxx.) Only one sysplex root is allowed for all systems participating in a shared file system.

The sysplex root is created by invoking the BPXISYZR job in SYS1.SAMPLIB.

After the job runs, the structure of a sysplex root file system would look like Figure 23 on page 160:

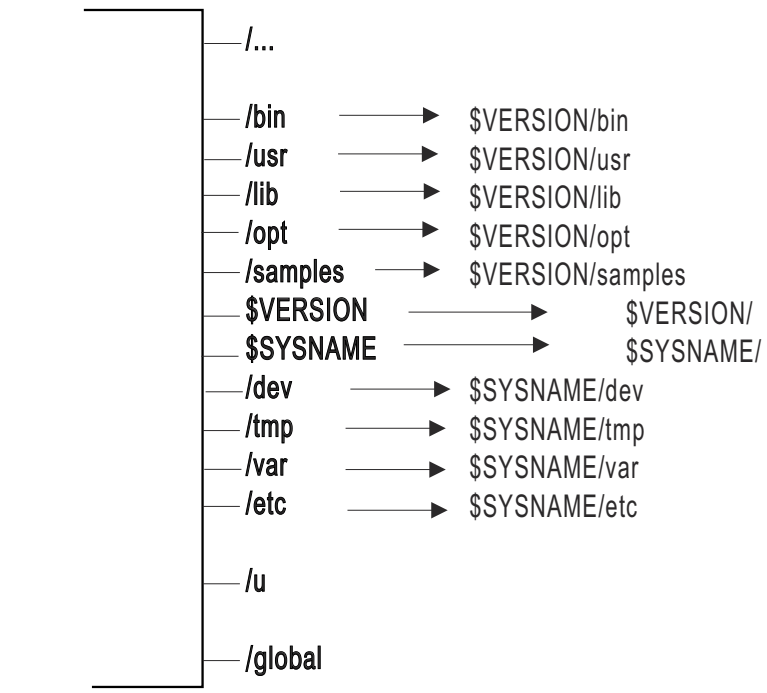


Figure 23. What the file system structure of a sysplex root looks like

No files or code reside in the sysplex root file system. It consists of directories and symbolic links only, and hence the size of the data set representing the sysplex root is very small.

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.

After you create the directories for each system-specific file system and the version file system, use the TSO UNMOUNT command to remount the sysplex root as read-only. Remounting the sysplex root file system as read-only prevents accidental corruption or full-file system problems with the sysplex root, both of which might require a sysplex IPL to recover. Additionally, most configurations will show improved performance if the file system is mounted as read-only. If a new directory needs to be added to the sysplex root file system, you can do the following tasks without disrupting the availability of the file system:

1. Use the TSO UNMOUNT command to remount the read-only file system to read/write mode.
2. Create the new directories.
3. Remount the file system in read-only mode.

Adding a system-specific or version file system to your shared file system configuration

In general, the contents of the sysplex root should only change when you need a new version file system or system-specific root file system directory for your shared file system configuration. When a system is IPLed (initialized), the mount processing for the sysplex root file system includes defining the appropriate $SYSNAME or $VERSION directory in the sysplex root file system if the sysplex root is mounted as read/write. Assuming that you have the sysplex root file system that is mounted as read-only, the procedure to use to create a new version file system or system-specific file system directory is as follows:

1. Use the following TSO command to change the file system from read-only to read/write mode:

```
UNMOUNT FILESYSTEM('Sysplex.Root.File.System.Name') REMOUNT(RDWR)
```

2. IPL the new system. When OMVS is initialized on the new system, the new directories are defined automatically. If the system is already active, you can manually define the appropriate directories by using ISHELL or the shell **mkdir** command.
3. To change the file back to read-only mode, use the following TSO command:

```
UNMOUNT FILESYSTEM('Sysplex.Root.File.System.Name') REMOUNT(READ)
```

Creating a system-specific file system

Directories in the system-specific file system are used as mount points, specifically for `/etc`, `/var`, `/tmp`, and `/dev`.

o create the system-specific file system, you need to run the appropriate 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 a file system.

After you invoke the job, the system-specific file system structure would look like Figure 24 on page 161:

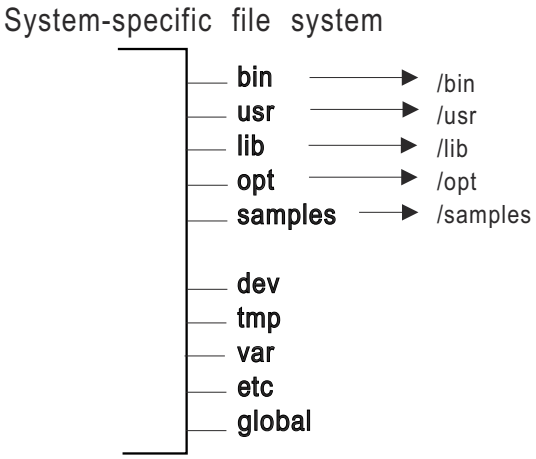


Figure 24. What the structure of a system-specific file system looks like

When you specify the MOUNT statements in the BPXPRMxx parmlib member for system-specific file systems, specify the UNMOUNT parameter. Then, if a system is removed from the sysplex, its file systems will be unmounted. If data sets are replaced during a reIPL, the new data sets are mounted as the original file systems.

Also, `/etc`, `/var`, `/tmp`, and `/dev` should be mounted similarly.

In order to use the &SYSNAME symbolic (defined in IEASYMxx) in the BPXPRMxx parmlib member, make sure that the name of the system-specific data set contains the system name as one of the qualifiers.

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE, loss of function might occur. For example, if the system-specific file

system mounted at /dev for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Mounting the version file system

The version file system is the IBM-supplied root file system. To avoid confusion with the sysplex root file system, *root file system* is called the *version file system*.

Figure 25 on page 162 shows a version file system.

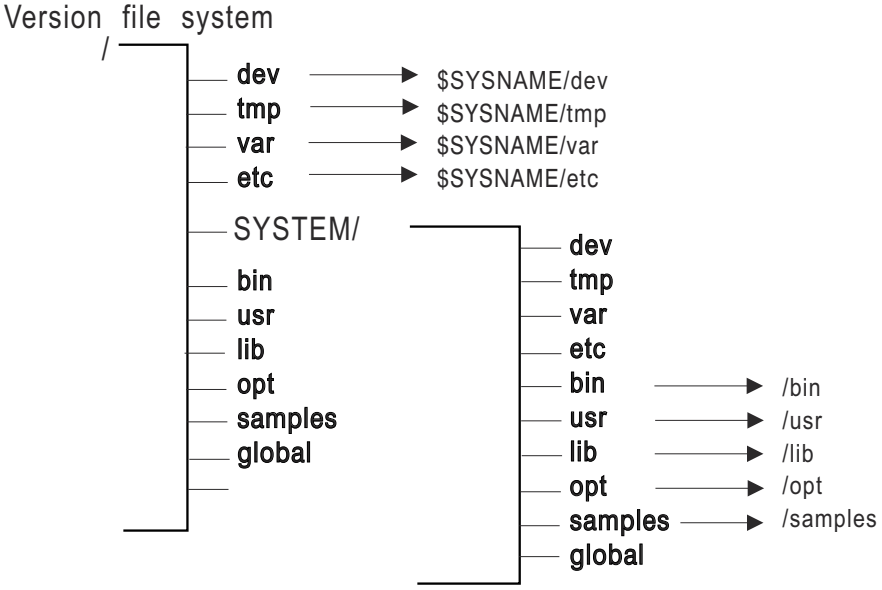


Figure 25. What a version file system looks like

When you are mounting the version file system, keep these guidelines in mind:

1. Mount the version file system read-only in a sysplex environment, and designate it AUTOMOVE. The mount point for the version file system is dynamically created if the VERSION statement is used in BPXPRMxx.
2. &SYSNAME as one of the qualifiers for the version file system data set name. In "Sysplex scenarios showing shared file system capability" on page 171, 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 file system in Portable Software Instance. CBPDO users obtain the version file system by following directions in the Program Directory. There is one version file system for each set of systems that are participating in a shared file system and who are at the same release level (that is, using the same SYSRES volume). In other words, each version file system denotes a different level of the system or a different service level. For example, if you have 20 systems that are participating in a shared file system, and 10 of those systems are at Release 9 and the other 10 are at Release 9A, then you'll have one version file system for the Release 9 systems and one for the Release 9A systems. In essence, you will have as many version file systems for the participating systems as you have different levels running.

Before you mount your version file system read-only, you might have some element-specific actions. These actions are described in "Post-installation actions for mounting the root file system in read-only mode" on page 121.

Automatically unmounting the version file system

The version file system can be unmounted automatically if the version auto-unmount attribute of the version file system is set prior to changing to a new version file system.

Note: A delay of 30 minutes will occur before the first attempt to unmount the unused version root.

To verify the value of the version auto-unmount attribute, issue one of the following commands:

- DISPLAY OMVS,FILES
- D OMVS,OPTIONS
- F BPXOINIT,FILESYS=DISPLAY

To set the version auto-unmount attribute for the current version file system, use parmlib statement VERSION=('*nnnn*',UNMOUNT) when IPLing with a new version file system, or use the statement with SET OMVS or SETOMVS VERSION=('*nnnn*',UNMOUNT).

When the file system with version auto-unmount attribute specified is no longer used as a version file system by any system in the sysplex, it will be unmounted automatically along with the file systems that are mounted under it. This unmount action will also be done during dead system takeover (member gone recovery) for a file system with the version auto-unmount attribute.

Restriction: All systems in the sysplex must be at least z/OS V2.3 or version auto-unmount processing will not be performed. To automatically unmount version file systems, perform these one-time actions once all systems in the sysplex are at z/OS V2.3 or later:

1. Change your BPXPRMxx file system member to specify VERSION('*nnnn*',UNMOUNT).
2. Use SET OMVS or SETOMVS to change any current version file system to UNMOUNT.

Creating a couple data set (CDS)

The TYPE(BPXMCDS) 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 TYPE(BPXMCDS) CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two couple data sets: 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 the CDS for z/OS UNIX System Services. The z/OS UNIX CDS controls shared file system mounts and will eventually contain information about all systems participating in the shared file system configuration.

   This system processes its BPXPRMxx parmlib member, including all its ROOT and MOUNT statement information. 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 file system.
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.

Following is the sample JCL with comments. The statements in bold contain the values that you specify based on your environment. Place the primary and alternate couple data sets on separate volumes.

```
//*
//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 z/OS UNIX CDS.
        Default =     8          */
        NOCATALOG
    /* Default is not to CATALOG */
    DATA TYPE(BPXMCDS)
    /* The type of data in the data set being created for OMVS.
```

```
     BPXMCDS is the TYPE for OMVS. */ITEM NAME(MOUNTS) NUMBER(100)
  /* Specifies the number of MOUNTS that can be supported by OMVS.*/
     Default =   100
     Suggested minimum =    10
     Suggested maximum = 35000  */
  ITEM NAME(AMTRULES) NUMBER(50)
   /* Specifies the number of automount rules that can be supported by OMVS.*/
      Default =    50
      Minimum =    50
      Maximum =  5000           */

    /* 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 namespecified on the volume specified.     */
    MAXSYSTEMS(8)
    /* Specifies the number of systems to be supported by the z/OS UNIX 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(100)
    /* Specifies the number of MOUNTS that can be supported by OMVS.
       Default =   100
       Suggested minimum =    10
       Suggested maximum = 35000 */
     ITEM NAME(AMTRULES) NUMBER(50)
    /* Specifies the number of automount rules that can be supported by OMVS.
       Default =     50
       Minimum =     50
       Maximum =   5000     */
```

AMTRULES specifies the number of automount rules that can be supported by z/OS UNIX. It must be large enough to describe each automount-managed directory in your automount policy. The maximum value is 5000. Generally, each managed directory requires one AMTRULE, and each generic or specific rule for that managed directory requires one-half of an AMTRULE. For example, an automount policy consisting of two managed directories where the first managed directory contains a single generic rule and the second managed directory contains three specific rules would require a total of four AMTRULES. The first managed directory requires 1.5 AMTRULES and the second managed directory requires 2.5 AMTRULES.

Important: 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 might have specified 1000 file systems to be automounted, but if you expect only 50 to be used concurrently, then factor these 50 into your MOUNTS value.

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. 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 z/OS UNIX CDS to XCF

Update the active COUPLExx parmlib member to define a primary and alternate z/OS UNIX couple data sets to XCF. Place the primary and alternate couple data sets on separate volumes. (The sample JCL in “Creating a couple data set (CDS)” on page 163 shows the primary CDS on volume 3390x1 and the secondary CDS on 3390x2.)

Figure 26 on page 165 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 26. COUPLExx parmlib member

The MVS operator commands (DISPLAY XCF, SETXCF, DUMP, CONFIG, and VARY) enable the operator to manage the z/OS UNIX CDS.

Customizing BPXPRMxx for a shared file system

You can use one BPXPRMxx member to define all the file systems in the sysplex. 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. The sharing is done by using system symbolics. Figure 29 on page 174 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 30 on page 175.

Tip: You can use the USS_CLIENT_MOUNTS check that is provided by IBM Health Checker for z/OS to verify that, in a shared file system configuration, sysplex-aware file systems are not function-shipping.

When setting up a shared file system in a sysplex, use the parameters that are described in Table 21 on page 165.

Table 21. Parameters used when setting up shared file systems in a sysplex. This table lists the parameters that are used when setting up shared file systems in a sysplex.

Parameter	What it does
SYSPLEX(YES)	Indicates that this system is to participate in the shared file system configuration. This involves joining the SYSBPX XCF group. Those systems that specify SYSPLEX(YES) make up the systems that participate in the shared file system configuration and are members of the SYSBPX XCF group. 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 ignores the SYSPLEX(YES) value and will initialize with SYSPLEX(NO). This system will not participate in a shared file system support after the initialization completes.

Table 21. Parameters used when setting up shared file systems in a sysplex. This table lists the parameters that are used when setting up shared file systems in a sysplex. *(continued)*

Parameter	What it does
VERSION('*nnnn*')	Allows multiple releases and service levels of the binaries to coexist and participate in a shared file system. *nnnn* is a qualifier to represent a level of the version file system. 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 file system. There is one version file system for every instance of the VERSION parameter. More information about version file system appears in "Mounting the version file system" on page 162.
The SYSNAME(*sysname*) parameter on the MOUNT statements	Specifies the particular system on which a mount is to be performed. *sysname* is a 1-to-8 alphanumeric name of the system. This system will then become the owner of the file system that is mounted. The owning system must also be IPLed with SYSPLEX(YES). **Tip:** Only specify a SYSNAME() value if you want only the specified system to be the file system owner. The MOUNT statement is ignored during z/OS UNIX initialization processing if SYSNAME() specifies another system. Once mounted on the specified owner, the file system will become locally available as a client or non-owner system. For SET OMVS and SETOMVS processing, the MOUNT statement is processed and the MOUNT is function-shipped to the system specified by SYSNAME(). If SYSNAME() is used with a value other than &SYSNAME. then there should not be any subsequent parmlib MOUNT statements that specify a MOUNTPOINT() with a path name that includes a directory in this file system The SYSNAME parameter is also used with SETOMVS when moving file systems, as demonstrated in "Moving file systems in a sysplex" on page 183.
The AUTOMOVE, NOAUTOMOVE, and UNMOUNT parameters on the ROOT and MOUNT statements	Indicate what happens to the file system if the system that owns that file system goes down. Note that the system list form of the AUTOMOVE parameter only applies to the MOUNT statement, and not to the ROOT statement. • AUTOMOVE without a system list specifies that ownership of the file system is automatically moved to another system. It is the default. • AUTOMOVE with a system list (SYSLIST) indicates which systems the file system should or should not be moved to when the owning system leaves the sysplex. • NOAUTOMOVE specifies that the file system will not be moved if the owning system goes down and the file system is not accessible. • UNMOUNT specifies that the file system will be unmounted when the system leaves the sysplex. This option is not available for automounted file systems. Define your version and sysplex root file system data as AUTOMOVE, and define your system-specific file systems as UNMOUNT. Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE. If you do, the file system that is defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted. To ensure that the root is always available, use the default, which is AUTOMOVE.

The owner of a file system is the first system that processes the mount. This system always accesses the file system locally; that is, the system does not access the file system through a remote system. Other non-owning systems in the sysplex access the file system either locally or through the remote owning system, depending on the PFS and the mount mode. If a PFS allows a file system to be locally accessed on all systems in a sysplex for a particular mode, then the PFS is *sysplex-aware* for that mount mode.

Even if a PFS is sysplex-aware for a particular mode, if a non-owning system does not have DASD connectivity, the file system is accessed remotely through the owning system. For example, a zFS that is mounted with the NORWSHARE setting is non-sysplex aware for read/write mode because all non-owning systems must access read/write file systems through the remote owning system. The non-owning systems are said to be *sysplex clients*. However, a zFS that is mounted with NORWSHARE in read-only mode or a zFS that is mounted with RWSHARE is sysplex-aware. Each system can access file systems locally and does not need to contact the owning system. For more information, see "File system availability" on page 181.

TFS file systems do not participate in move operations, regardless of the AUTOMOVE setting. They are unmounted if the file system owner becomes unavailable.

Restrictions: Keep the following restrictions in mind:

1. An AUTOMOVE file system cannot be moved to a system where z/OS UNIX was shut down or where F BPXOINIT,SHUTDOWN=*fileowner* was issued.
2. Automount-managed file systems are handled as AUTOMOVE if the file system is being locally used.
3. NOAUTOMOVE and system list are permitted for a sysplex-aware file system and are honored on a V1R9 system. However, if the file system is moved to a system at an earlier release, the automove setting is changed to AUTOMOVE.

Table 22 on page 167 shows what happens during soft shutdown for various AUTOMOVE settings. Soft shutdown is done by issuing one of the following MODIFY commands:

```
F BPXOINIT,SHUTDOWN=FILESYS
F BPXOINIT,SHTUDOWN=FILEOWNER
```

A *leaf file system* refers to a file system that does not contain any file systems that are mounted on a directory within that file system. A *subtree* is the file system and all file systems that are mounted beneath that file system.

Table 22. Soft shutdown actions for various AUTOMOVE settings. This table lists the soft shutdown actions for various AUTOMOVE settings.

Automove value	Action taken
NOAUTOMOVE or UNMOUNT	An attempt to unmount the file system occurs. The unmount fails if it is not a leaf file system.
AUTOMOVE without a system list	Moves the file system to any system. If the move fails, the unmount is not attempted.
AUTOMOVE with a system list	Moves the file system to any system. If the file system cannot be moved, then the unmount is not attempted.

Table 23 on page 168 shows what happens during soft shutdown for various AUTOMOVE settings for an OMVS shutdown, performed by using the F OMVS,SHUTDOWN system command.

Table 23. OMVS shutdown actions for various AUTOMOVE settings. This table lists the shutdown actions that are taken by OMVS for various AUTOMOVE settings.

Automove value	Action taken
NOAUTOMOVE	An attempt to unmount the file system occurs. The unmount fails if it is not a leaf file system and the file system becomes unowned. 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.
UNMOUNT	The file system is unmounted, and all the file systems that are mounted within it are also unmounted.
AUTOMOVE without a system list	Moves the file system to any system. If the move fails, the file system becomes unowned. The file system remains unowned until the last owning system restarts or until the unowned recovery daemon can establish a new file system owner.
AUTOMOVE with a system list	An attempt to move ownership of the file system to eligible systems (as defined by the INCLUDE or EXCLUDE system list) is performed. If no systems could become the file system owner, the file system is unmounted, as well as all the file systems mounted within it.

Automount-managed file systems are unmounted by a soft shutdown operation if the file system is not referenced by any other system in the sysplex. If it is referenced by another system or systems, ownership of the file system is moved. If the move fails, an unmount is not attempted and ownership does not change.

Table 24 on page 168 shows what happens during dead system takeover for various AUTOMOVE settings for sysplex-aware file systems. *Dead system takeover* (otherwise known as Member Gone Recovery) is the action that is taken by systems in a sysplex when they attempt to take over ownership of file systems that were previously owned by a system that has just left the XCF BPXGRP member group.

Table 24. Dead system (member gone) takeover for various AUTOMOVE settings. This table lists the action that is taken for dead system takeover for various AUTOMOVE settings.

Automove value	Action taken
NOAUTOMOVE	The file system becomes unowned. 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 mount point for the file system is still in use.
UNMOUNT	The file system is unmounted, and all the file systems that are mounted within it are also unmounted.
AUTOMOVE without a system list	An attempt to move ownership of the file system to all other eligible systems in the participating group is performed. If another file system cannot become the owner of the file system, the file system becomes unowned. The file system remains unowned until the last owning system restarts or until the unowned recovery daemon can establish a new file system owner.

Table 24. Dead system (member gone) takeover for various AUTOMOVE settings. This table lists the action that is taken for dead system takeover for various AUTOMOVE settings. *(continued)*	
Automove value	**Action taken**
AUTOMOVE with a system list	An attempt to move ownership of the file system to eligible systems (as defined by the INCLUDE or EXCLUDE system list) is performed. If another system could not become the file system owner, the file system is unmounted, in addition to all the file systems mounted within it.

There is no attempt to take over automount-managed file systems if the file system is not being used locally. Automount-managed, unowned file systems are unmounted.

Table 25 on page 169 shows what happens during PFS termination for various AUTOMOVE settings.

Table 25. PFS termination for various AUTOMOVE settings. This table discusses what happens if PFS is terminated	
What happens if . . .	**Action taken**
NOAUTOMOVE	The file system is unmounted, and all the file systems that are mounted within it are also unmounted.
UNMOUNT	The file system is unmounted, and all the file systems that are mounted within it are also unmounted.
AUTOMOVE without a system list	An attempt to move ownership of the file system to all other eligible systems in the participating group is performed. If another system cannot become the owner of the file system, the file system is unmounted, in addition to all the file systems mounted within it.
AUTOMOVE with a system list	An attempt to move ownership of the file system to eligible systems (as defined by the INCLUDE or EXCLUDE system list) is performed. If another system cannot become the file system owner, the file system is unmounted, in addition to all the file systems mounted within it.

Table 26 on page 169 shows what happens when a move file system is requested to move a specific file system to any target system (wildcard is used). A move file system request can be issued with a SETOMVS operator command or a `chmount` shell command.

Table 26. Move a specific file system to any system for various AUTOMOVE settings. This table lists what happens during an automove for a fixed file system.	
What happens if . . .	**For sysplex-aware file systems**
NOAUTOMOVE	Move is attempted to all systems.
UNMOUNT	Move is attempted to all systems.
AUTOMOVE without a system list	Move is attempted to all systems.
AUTOMOVE with a system list	Move is attempted only to systems in the system list.

Table 27 on page 170 shows what happens when a move filesystem is requested to do a multi-file system move, moving all file systems from a system to a specific target system. A move file system request can be issued with a SETOMVS operator command or a `chmount` shell command.

Table 27. Move all file systems from a system to a specific target system for various AUTOMOVE settings. This table lists what happens when all file systems are moved to a specific target system.	
What happens if . . .	**Action taken**
NOAUTOMOVE	Move is not attempted.
UNMOUNT	Move is not attempted.
AUTOMOVE without a system list	Move is attempted to the target system.
AUTOMOVE with a system list	Move is attempted to the target system and the system list is ignored.

Rules: Define your version and sysplex root file system as AUTOMOVE. Also:

1. Define your system-specific file systems as UNMOUNT.
2. Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system under it as AUTOMOVE. If you do, the file system that is defined as AUTOMOVE will not be recovered after a system failure until that failing system is restarted.

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

1. If a file system that is mounted as AUTOMOVE with or without a SYSLIST is not moved or recovered as expected, use D OMVS,MF on all systems to review MOUNT or MOVE failures relating to the specific file system.

Using system lists

When mounting file systems in the sysplex, you can specify a prioritized system list to indicate where the file system should or should not be moved to when the owning system leaves the sysplex changes due to any of the following situations:

- A soft shutdown request was issued.
- A dead system takeover took place (when a system leaves the sysplex without a prior soft shutdown).
- A PFS terminates on the owning system.
- A request to move ownership of the file system was issued.

There are different ways to specify the system list.

- On the MOUNT statement in BPXPRMxx, specify the automove keyword, including the indicator and system list.
- For the TSO MOUNT command, specify the automove keyword, including the indicator and system list.
- Use the `mount` shell command.
- Use the ISHELL MOUNT interface.
- Specify the MNTE_SYSLIST variable for REXX.
- Specify the indicator and system list for the AUTOMOVE option in the `chmount` shell command.
- Specify the indicator and system list for the AUTOMOVE option in the SETOMVS operator command.

Using wildcards

When you specify the system list, you can use wildcards in certain situations.

You can use the wildcard support on all methods of mounts, including the MOUNT statement in BPXPRMxx, the TSO MOUNT command, the `mount` shell command, the ISHELL MOUNT interface, the MNTE_SYSLIST variable for REXX, C program, and assembler program.

Restriction: The wildcard is only allowed in an INCLUDE list. It is not allowed in an EXCLUDE list. Also, it must be the last item (or the only item) in the system list.

Example

If you have many systems in your sysplex, you can specify only the systems that should have priority and use a wildcard to indicate the rest of the systems.

```
AUTOMOVE INCLUDE(s1,S2,...*)
```

At first glance, AUTOMOVE INCLUDE (*) appears to work the same way as AUTOMOVE because all of the systems will try to take over the file system. However with AUTOMOVE INCLUDE (*), if none of the systems can take over the file system, it is unmounted. If AUTOMOVE is used, the file system remains mounted but becomes unowned.

zFS sysplex considerations when using system lists

zFS file systems in a mixed zFS sysplex-aware and non-sysplex aware configuration typically cannot be moved from a zFS sysplex-aware system to a zFS non-sysplex aware system. For more information, see Using zFS in a shared file system environment in *z/OS File System Administration*.

Sysplex scenarios showing shared file system capability

The BPXPRMxx member of SYS1.PARMLIB needs to reflect the requirements for the sysplex configuration. Update it according to your needs.

Scenario 1: First system in the sysplex

Figure 27 on page 172 and Figure 28 on page 173 show a z/OS UNIX file system configuration for a shared file system. SYSPLEX(YES) and a value on VERSION are specified, and a directory is dynamically created on which the version file system data set is mounted. This type of configuration requires a sysplex root and system-specific file system.

After you create the directories for each system-specific file system and the version file system, use the TSO UNMOUNT command to remount the sysplex root as read-only. Remounting the sysplex root file system as read-only prevents accidental corruption or full-file system problems with the sysplex root, both of which might require a sysplex IPL to recover. Additionally, most configurations will show improved performance if the file system is mounted as read-only. If a new directory needs to be added to the sysplex root file system, you can do the following tasks without disrupting the availability of the file system:

1. Use the TSO UNMOUNT command to remount the read-only file system to read/write mode.
2. Create the new directories.
3. Remount the file system in read-only mode.

```
BPXPRMxx for (SY1)

FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')                1
TYPE(ZFS) MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')                   2
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.ZFS')         3
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME')

MOUNT
FILESYSTEM('OMVS.ROOT.ZFS')                     4
TYPE(ZFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')                5
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME/dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')                6
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME/tmp')
.
.
.
```

Figure 27. BPXPRMxx setup — sharing file systems

1 This is the sysplex root file system and was created by running the BPXISYZR job.

2 This file system contains configuration files and symbolic links that will be useful across all the systems in the file sharing environment. There is only one such file system for the file sharing environment, hence AUTOMOVE is used for availability reasons. The first system's BPXPRMxx member that specifies this MOUNT statement will mount the file system. As other systems enter the file sharing environment and their BPXPRMxx members specify the same MOUNT statement, this statement is ignored because that file system has been mounted on the same mount point.

3 This is the system-specific file system, and was created by running the BPXISYZS job. UNMOUNT is specified because this file system is system-specific and ownership of the file system should not move to another system if 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.

4 This is the previous root file system (version file system). It should be mounted read-only. Its mount point is created dynamically and the name of the file system 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.

5 This file system contains the system-specific /dev information. UNMOUNT 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.

6 This file system contains system-specific /tmp information. UNMOUNT is specified because this file system is system-specific; ownership should not move to another system if the owning system

goes down. The MOUNTPOINT statement /&SYSNAME./tmp will resolve to /SY1/tmp during parmlib processing.

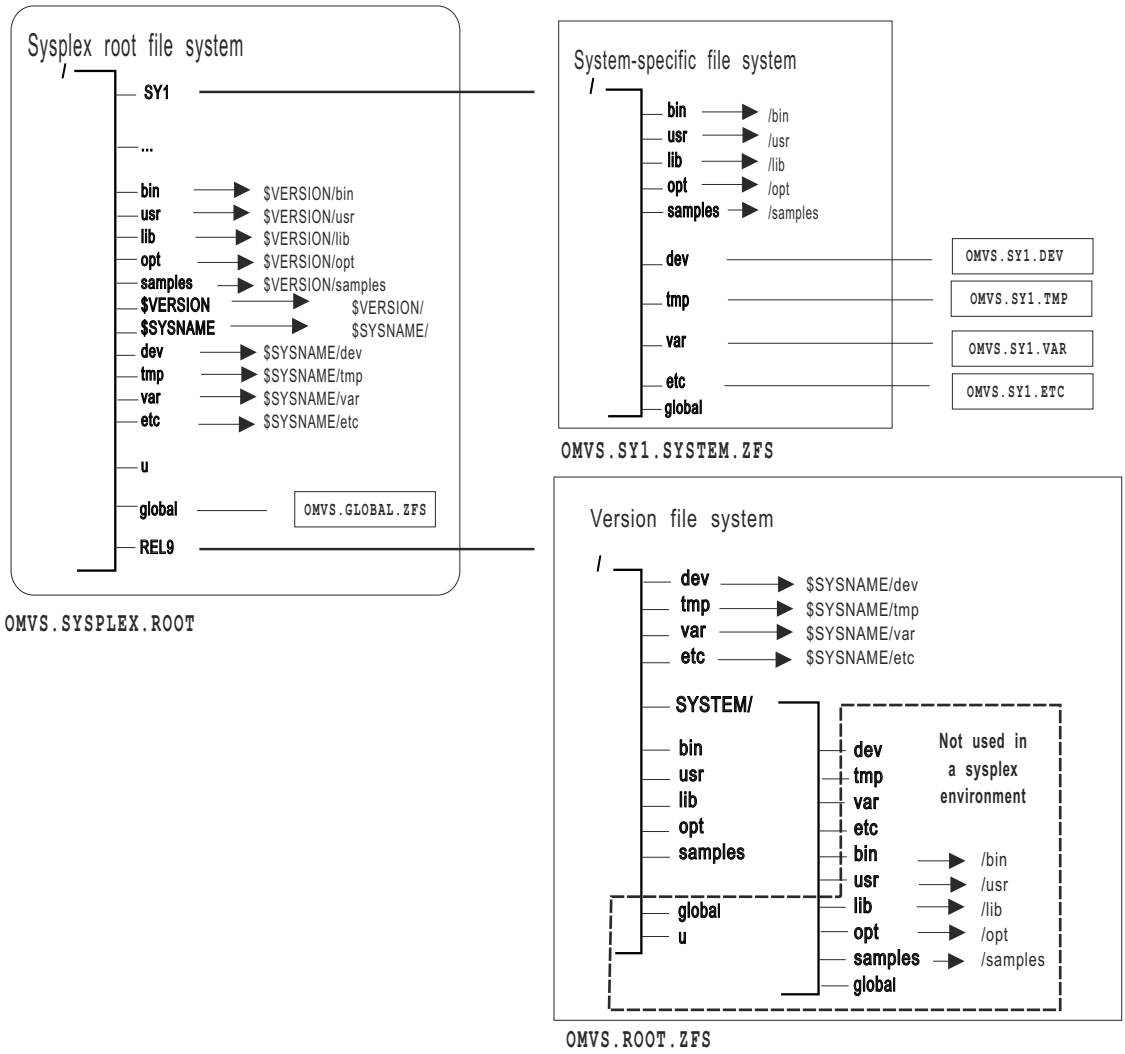


Figure 28. Shared file systems in a sysplex

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

- If you specify 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 specify 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 previous 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 path name /REL9/bin. The contents of /REL9/bin will now be displayed.

Scenario 2: Multiple systems in the sysplex using the same release level

Figure 31 on page 176 shows another SYSPLEX(YES) configuration. In this configuration, however, two or more systems are sharing the same version file system (the same release level of code). Figure 29 on

page 174 shows a sample BPXPRMxx for the entire sysplex (what IBM suggests) using &SYSNAME. as a symbolic name, and Figure 30 on page 175 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(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(ZFS) MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE

MOUNT FILESYSTEM('OMVS.USER.ZFS'
MOUNTPOINT('/u') AUTOMOVE
TYPE(ZFS) MODE(RDWR)

MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.ZFS')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME')

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

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME/dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME/tmp')
.
.
.
```

Figure 29. Sharing file systems: one version file system and one BPXPRMxx for the entire sysplex

BPXPRMS1 (for SY1)

```
FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(ZFS) MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE

MOUNT
FILESYSTEM('OMVS.SY1.SYSTEM.ZFS')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/SY1')

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

MOUNT FILESYSTEM('OMVS.SY1.DEV')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/SY1/dev')

MOUNT FILESYSTEM('OMVS.SY1.TMP')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/SY1/tmp')
.
.
.
```

BPXPRMS2 (for SY2)

```
FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(ZFS) MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE

MOUNT
FILESYSTEM('OMVS.SY2.SYSTEM.ZFS')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/SY2')

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

MOUNT FILESYSTEM('OMVS.SY2.DEV')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/SY2/dev')

MOUNT FILESYSTEM('OMVS.SY2.TMP')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/SY2/tmp')
```

Figure 30. Sharing file systems: one version file system and separate BPXPRMxx members for each system in the sysplex

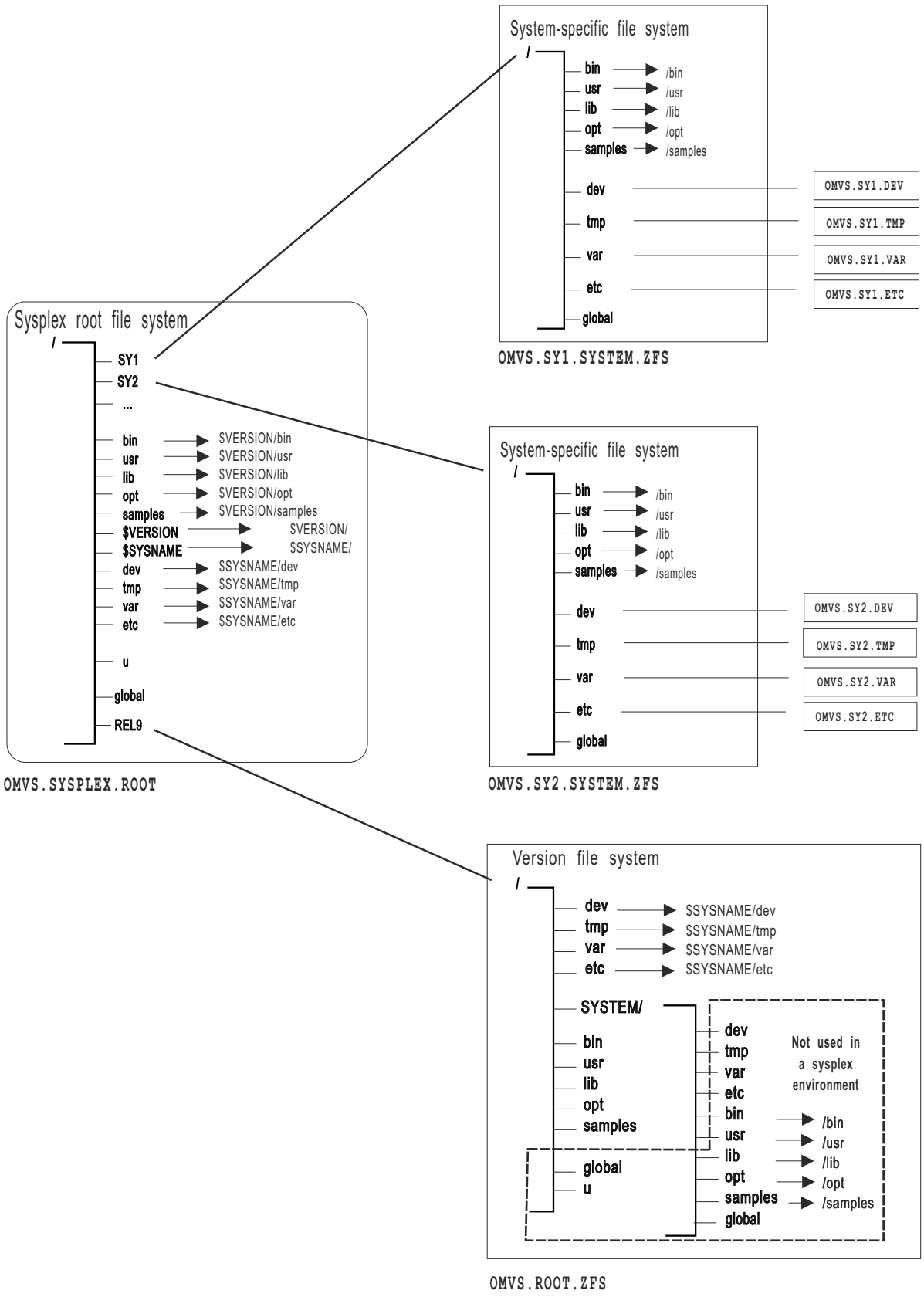


Figure 31. Sharing file systems in a sysplex: multiple systems in a sysplex using the same release level

In this scenario, where multiple systems in the sysplex are using the same version file system, 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 path name /REL9/bin. The contents of this directory are displayed.

Scenario 3: Multiple systems in a sysplex that use different release levels

If your participating group is in a sysplex that runs multiple levels of z/OS, your configuration might look like the one in Figure 33 on page 178. Because each system is running a different level of z/OS, each system has different version file system data sets.

- OMVS.SYSR9A.ROOT.ZFS is the version file system on SYS1.
- OMVS.SYSR9.ROOT.ZFS is the version file system on SYS2.

Figure 32 on page 177 shows two BPXPRMxx parmlib members that define the file systems in this configuration. Figure 34 on page 179 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 file system data sets.

```
BPXPRMxx (for SY1)

FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

VERSION('REL9A')
SYSPLEX(YES)

ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(ZFS)  MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE
AUTOMOVE

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.ZFS')
TYPE(ZFS)  MODE(RDWR) UNMOUNT PARM('NORWSHARE')
PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME')

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

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(ZFS)  MODE(RDWR) UNMOUNT PARM('NORWSHARE')
PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME/dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(ZFS)  MODE(RDWR) UNMOUNT PARM('NORWSHARE')
PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME/tmp')
.
.
.
```

```
BPXPRMxx (for SY2)

FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
ASNAME(ZFS)

VERSION('REL9')
SYSPLEX(YES)

ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE(ZFS)  MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.ZFS')
TYPE(ZFS)  MODE(RDWR) UNMOUNT

MOUNTPOINT('/&SYSNAME')

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

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(ZFS)  MODE(RDWR) UNMOUNT

MOUNTPOINT('/&SYSNAME/dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(ZFS)  MODE(RDWR) UNMOUNT

MOUNTPOINT('/&SYSNAME/tmp')
```

Figure 32. BPXPRMxx setup for multiple systems that share file systems with different release levels

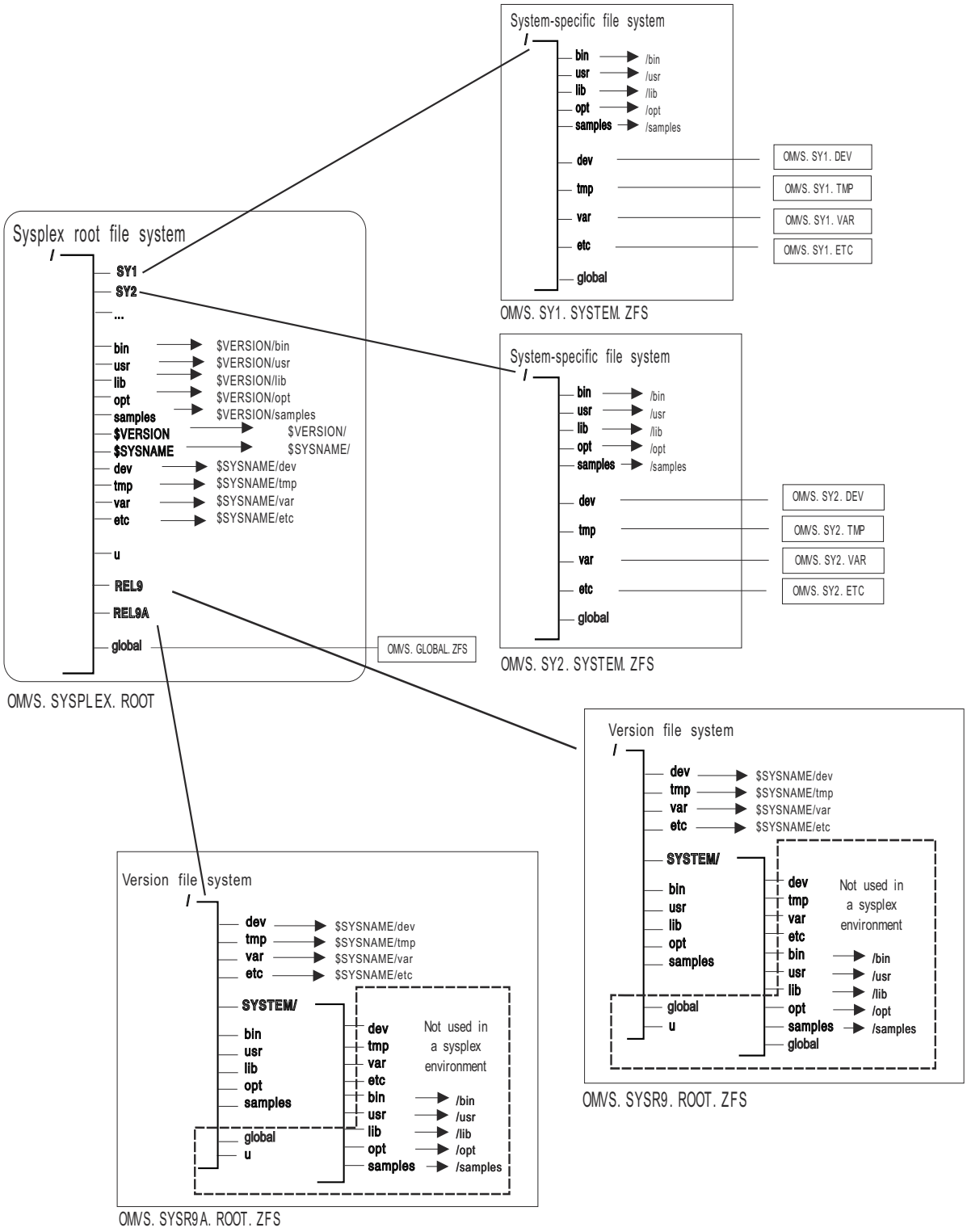


Figure 33. Sharing file systems between multiple systems that uses different release levels

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 that points to `$VERSION/bin`, the symbolic link must be resolved first. $VERSION resolves to /SYSR9A on SY1, which makes the path name `/SYSR9A/bin`. The contents of this directory is now displayed. If `ls -l /bin/` is issued on SY2, the contents of `/SYSR9/bin` is displayed.

From SY2, you can display information on SY1 by fully qualifying the directory.

Example: To view SY1's /bin/ directory:

```
ls -l /SY1/bin/
```

```
One BPXPRMxx member to define file systems for the entire sysplex
using different releases

FILESYSTYPE
TYPE(ZFS)
ENTRYPOINT(IOEFSCM)
PARM(' ')

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

ROOT
FILESYSTEM ('OMVS.SYSPLEX.ROOT')
TYPE(ZFS) MODE(READ)


MOUNT
FILESYSTEM('OMVS.GLOBAL.ZFS')
TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/global') AUTOMOVE

MOUNT FILESYSTEM('OMVS.USER.ZFS')
MOUNTPOINT('u') AUTOMOVE
TYPE(ZFS) MODE(READ)

MOUNTFILESYSTEM('OMVS.&SYSNAME..SYSTEM.ZFS')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME.')

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

MOUNT
FILESYSTEM('OMVS.&SYSNAME..DEV')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME./dev')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..TMP')
TYPE(ZFS) MODE(RDWR) UNMOUNT PARM('NORWSHARE')
MOUNTPOINT('/&SYSNAME./tmp')
.
.
.
```

Figure 34. One BPXPRMxx parmlib member for multiple systems that shares file systems and uses different release levels

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

Using the /global directory

The /global directory in the sysplex root is intended to hold configuration and data (non-executable) files that have a sysplex scope. Elements and licensed programs that need to have a single and consistent way to access configuration and data files across a sysplex will provide instructions or scripts to create files in /global. Because these scripts are also applicable in non-sysplex environments, a /global directory also exists in the root file system.

Tip: To keep the sysplex root small and free of configuration and data files, it is highly recommended that a file system be created and mounted on the /global directory.

Customers can also use the /global directory to store local customization files. One such example is the automount policy. To understand how the /global directory can be used to provide a single and consistent automount policy across the sysplex, see Chapter 6, “Using the automount facility,” on page 147 .

Like configuration files, /global can also be used to hold symbolic links (symlinks) that can act as shortcuts for potentially long path names. Those shortcuts are a convenient and consolidated way to access multiple levels of a licensed program. For example, licensed programs that are shipped independent of z/OS reside in /usr/lpp... and might not be necessarily installed on every system in the sysplex. The /global directory can be used as a common repository for different levels of that licensed program.

Tip: A customer wants to create a convenient method to access the different levels of Java™ that are available across the various systems in the sysplex. In this scenario, three levels of Java are installed on two different systems in the sysplex. For example, Java 6 is installed on the z/OS V2.1 system and both Java 7 and Java 8 are installed on the z/OS V2.2 system.

The following assumptions are made:

- A /globaldirectory exists in the sysplex root.
- A file system is mounted on /global.
- Permission bits for /global were changed to 7,5,5 after the file system was mounted.

To create a common repository for different levels of Java, follow these steps:

1. Create the /global/java directory.
2. Create the /global/java/java6 symbolic link to the directory where Java 6 is installed.
3. 3. Create additional symbolic links as necessary to represent additional levels of Java.

For example:

```
/global/java/java6  --symlink→  /zOS21/usr/lpp/java6
/global/java/java7  --symlink→  /zOS22/usr/lpp/java7
/global/java/java8  --symlink→  /zOS22/usr/lpp/java8
```

Tip: To eliminate the need for the java/ directory in the preceding example, you can create symbolic links directly under /global directory.

Using the automount policy

The default delay time for automount is 10. Do not use a value less than 10. To verify the setting of the delay time, use the USS_AUTOMOUNT_DELAY check that is provided by IBM Health Checker for z/OS.

Rule: You must keep the automount policy consistent across all the participating systems in the sysplex. The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

Your automount policy most likely resided in the /etc/auto.master and /etc/u.map files. For those using shared file systems, each participating system has a separate /etc file system. In order for the 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.

AUTOMOUNT is the preferred method of managing the /u directory. You do not need a mount statement for /u in the BPXPRMxx parmlib member.

For example, both SY1 and SY2 have the following files:

- /etc/auto.master

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

- /etc/u.map

```
name       *
type       ZFS
filesystem OMVS.<uc_name>.ZFS
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 that is 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 that is specified in /etc/u.map and the values for other parameters in /etc/u.map and /etc/auto.master.

An alternate way of maintaining a consistent automount policy across systems in the file sharing environment is to place the respective files in a common repository. The /global directory in the sysplex root and the file system that has been mounted at this mount point can be used for such a purpose.

Assuming the permission bits of /global have been changed to 7,5,5 after mounting the file system, the following steps can be used.

1. Create a file called /global/auto.master. This file would be equivalent to the /etc/auto.master file.
2. Create /global/u.map file and place the automount policy in this file. This file would be equivalent to the /etc/u.map file.
3. Then, on every system participating in a shared file system, create a symbolic link for /etc/auto.master to /global/auto.master. Also delete each system-specific /etc/u.map file.

This now allows for a single automount policy to reside in a single location in the file sharing environment.

File system availability

In the shared file system environment, file system availability and accessibility depend on a number of important factors. These factors can vary depending on how a file system is mounted and the capability of the file system to manage itself in a sysplex environment. After you set up the shared file system environment for cross-system communication, it will be helpful to understand how file system availability is provided to your systems, and what kinds of actions can cause interruptions to that availability.

Minimum setup required for file system availability

For DASD file systems, at least one system in the shared file system group needs to have a physical I/O path to the volume where the file system resides and the volume is varied online. Without connectivity from at least one system, the file system will not be available to any of the systems in the shared file system group. Connectivity from one system can provide shared file system accessibility to the file system for all other systems in the shared file system group.

All systems need to have the physical file system (PFS) started. Accomplish this by placing the appropriate FILESYSTYPE statement in the BPXPRMxx parmlib member that is used in the configuration. Additionally, any necessary subsystems required by the PFS must be started and configured, especially if this system is to function as the file system owner. For example, the NFS Client PFS requires that the TCP/IP subsystem be started and a network connection configured.

Read/write connections for non-sysplex aware file systems

Most physical file systems (PFSs) allow only one connection for update at a time. Such file systems are called *non-sysplex aware for update*. This is directly related to the mount mode of the file system. With zFS, for example, only one system can connect to the file system with a mode of RDWR. That system is called the *file system owner*.

The other systems that want to participate in shared file systems will also request an RDWR mount, but their access will be provided via cross-system messaging with the file system owner, which has already established the read/write connection. These systems are called *file system clients*. When the file system owner becomes unavailable (for example, through system shutdown), it will be important for another system (one of the file system clients) to have the file system volume varied online so that a new owner can be established. This helps ensure maximum file system availability in the shared file system group.

Read-write connections for sysplex-aware file systems

Some PFSes can handle multiple concurrent connections for update. They can manage the serialization of such requests. Such file systems are called *sysplex aware for update*. Most network file systems have this capability. NFS Client is one such file system type.

For a read/write mount to NFS Client, each system in the shared file system group will make a direct connection to NFS. The first system to make such a connection is still called the file system owner. All subsequent systems to make a direct connection are considered non-owners, rather than clients. This type of multiple direct connection for read/write access allows for maximum I/O performance by eliminating the need to send requests to the file system owner.

However, sometimes a non-owning system cannot make a direct connection to the PFS even after meeting the minimum requirements (for example, sometimes requests to NFS Client time out before they are satisfied). That system might be given a cross-system messaging connection, making it a client to the file system. While this is not the optimal mount mode for this type of file system, it does allow access to the file system.

zFS provides the capability for sysplex-aware for update. This support is configurable and can provide the following levels of support:

No support for sysplex-aware for update
: zFS only allows a single system to connect to the file system for update. All other systems will function as client systems and will function-ship requests to the file system owner system.

Global support for sysplex-aware for update
: zFS supports multiple concurrent connections for update for all file systems.

Per file system support for sysplex-aware for update
: This level of support requires z/OS UNIX APAR OA29712 and zFS APAR OA29619. It provides the capability for specific file systems to be configured as sysplex-aware or non-sysplex aware for update. For more information, see Specifying zFS file systems as sysplex-aware in *z/OS File System Administration*.

Read-only connections for non-sysplex aware file systems

Some physical file systems such as TFS do not support multiple concurrent connections for read-only access. These are called *non-sysplex aware for read-only*, and are handled the same as the read/write connections for non-sysplex aware file systems.

Read-only connections for sysplex-aware file systems

Physical file systems that support multiple concurrent connections for read-only access are called *sysplex aware for read-only*. The zFS physical file system falls into this category. Such file systems are handled the same as the read/write connections for sysplex aware file systems. The read-only connections are attempted locally for each system in the shared file system group, but if the file system volume is not online to a system, then the system becomes a client to the file system by means of cross-system messaging with the owner.

Situations that can interrupt availability

Some situations might cause interruptions to file system availability on one or more systems. Following is a list of some of the most common causes. It is not meant to be an exhaustive list.

Loss of the file system owner
If the file system owner leaves the shared file system group (through system failure, soft shutdown, VARY, XCF, OFFLINE, or some other means), an attempt might be made to establish another file system owner if requested by the AUTOMOVE specification of the mount. If a new file system owner cannot be established, the file system will become unowned. It will be unavailable until the original owner can reclaim it, or until another owner is established through subsequent automated recovery actions performed by shared file system.

PFS termination
If a PFS terminates on one system, it can affect file system availability on other systems. If a PFS terminates on one system, any file systems of that type that are owned by other systems are not affected. File systems of that type are moved to new owners whenever possible if they are owned by the system where the PFS is terminating and if they can be automoved. These file systems remain accessible to other systems. If they cannot be moved to new owners, they are unmounted across the sysplex. It might not be possible to move a file system due to a lack of connectivity from other systems, or if the file system containing the mount point for the file system needed to be moved but could not be.

VARY volume,OFFLINE
When the volume for a file system is varied offline, it will make that file system inaccessible to that system. However, if the volume is online to other systems, it might still be accessible to those systems and to other systems via cross-system messaging. This would be the case for sysplex-aware file systems for read/write or read-only access. Unlike loss of the file system owner, varying a file system volume offline will not result in any attempt by the system to restore accessibility to systems on which it is lost.

Moving file systems in a sysplex

You might need to change ownership of the file system for recovery or reIPLing.

If you are working with file systems in a sysplex, consider these tips:

- To check for file systems that have already been mounted, use the **df** command from the shell.
- To move a file system in a sysplex, use the SETOMVS command that is used with the FILESYS, FILESYSTEM, mount point, and SYSNAME parameters. You can also use the **chmount** command from the shell. However, do not move system-specific file systems.

These examples assume you are working with file systems in a sysplex.

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.ZFS',SYSNAME=SY2
```

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

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

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function might occur. For example, if the system-specific file system mounted at /dev for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Also, opened FIFO files are automatically closed before the file system containing the FIFO is moved. They are closed because the in-storage FIFO data on the old system is not moved and is no longer accessible on new owning system.

Moving file systems to an earlier system

If you move a file system to an earlier system, existing NFS client connections to the files in that file system might be broken if Share Reservations are used. With Share Reservations, remote NFS clients can open files in such a way that no one else can open that file until the first program finishes and closes the file. For more information about Share Reservations, see v_open (BPX1VOP, BPX4VOP) — Open or create a file in *z/OS UNIX System Services File System Interface Reference*.

Share Reservations that attempt to deny reading or writing for files in a read-only file system are accepted but will not be enforced.

File systems cannot be remounted from read/write to read-only or from read-only to read/write while there are Share Reservations established on any file in that file system.

Restriction: A file system cannot be moved to an earlier system while there are active Share Reservations on any file in the file system. You will have to move the file system to a sysplex member at the z/OS V1R7 release level or later. Alternatively, you can stop the applications at the NFS clients who have put reservations on the files, or wait for them to finish.

If an NFS client is going to open a file that has Share Reservations set:

- That file must be owned by a system at the z/OS V1R7 level or higher before it can be opened.
- If the file is owned by a remote system that supports Share Reservations, they are enforced at the owner for all opens within the sysplex.
- If the file is owned by a remote system at a lower level, the client's open will fail. The reason code of the failure indicates that the file system must be moved to a sysplex member that is at the z/OS V1R7 release or later.

If the system goes down and there are Share Reservations on a file that is owned by a remote system:

1. If the file system is taken over by another z/OS V1R7 or later system, the reservations are reestablished at the new owner and enforced there.
2. If the file system is taken over by an owner that does not support Share Reservations, the NFS client's open is invalidated and subsequent operations from that client for this open are rejected. If you move the file system to a sysplex member that supports Share Reservations, the file can be reopened as it was before. You can use the AUTOMOVE parameter of the MOUNT command to restrict these takeovers to the systems that do support Share Reservations.

zFS sysplex considerations when moving file systems

zFS file systems in a mixed zFS sysplex-aware and non-sysplex aware configuration typically cannot be moved from a zFS sysplex-aware system to a zFS non-sysplex aware system. For more information, see Using zFS in a shared file system environment in *z/OS File System Administration*.

Implications of shared file systems during system failures and recovery

File system recovery in a shared file system environment takes into consideration file system characteristics such as the PFS capabilities, whether the file system is automount-managed, and the AUTOMOVE value. “Customizing BPXPRMxx for a shared file system” on page 165 describes the various AUTOMOVE values and the system actions taken for the various shutdown and recovery flows.

Generally, when an owning system fails, ownership of a file system that is mounted as AUTOMOVE is moved to another system and the file system remains usable. However, if a file system is mounted as read/write and the owning system fails, then all file system operations for files in that file system will fail, no matter what the value set with AUTOMOVE is. The failure 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. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information about BPX1CLO and BPX1OPN.

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

In some situations, even though a file system is mounted as AUTOMOVE, ownership of the file system might not be immediately moved to another system. This can 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 file system support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared file system. If a subsequent recovery attempt succeeds, the file system moves 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 as NOAUTOMOVE becomes unowned when the file system owner exits the sysplex. The file system remains unowned until the original owning system restarts or until the unowned file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use. File systems that are mounted under a NOAUTOMOVE file system will not be accessible via path name when the NOAUTOMOVE file system becomes available.

Restriction: Do not 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 NOAUTOMOVE file system becomes unowned, it will not be possible to perform path name lookup through it to the file systems mounted within it, which will make those file systems unavailable for new access. When ownership is restored to the unowned file system, access to the file systems mounted within it is also restored.

If a file system that is mounted as AUTOMOVE with or without a SYSLIST is not moved or recovered as expected, use D OMVS,MF on all systems to review MOUNT or MOVE failures relating to the specific file system.

Managing the 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 that you can take to help manage the placement of data if a system failure occurs.

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 that are provided by the various mount services to specify the original owner and subsequent owners for a particular file system.

“Customizing BPXPRMxx for a shared file system” on page 165 describes the behavior of the various AUTOMOVE options.

Table 28 on page 186 shows the AUTOMOVE options that you can use with the MOUNT command to manage file systems.

Table 28. AUTOMOVE options supported by the MOUNT command . This table lists the automove options that are supported by the MOUNT command.

Automove option	Action
UNMOUNT	Attempts are not made to keep the file system active when the current owner fails. The file system is unmounted when the owner is no longer active in the participating group, as well as all the file systems mounted within it. Use UNMOUNT when mounting system-specific file systems, such as those that would be mounted at `/etc`, `/dev`, `/tmp`, and `/var`.
NOAUTOMOVE	Attempts are not made to keep the file system active when the current owner fails. The file system remains 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 mount point for the file system 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 without a system list	Recovery of the file system is performed when the current owner fails. Use this option on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.
AUTOMOVE with a system list	AUTOMOVE(EXCLUDE|INCLUDE,sysname1,sysname2,...sysnameN) specifies managed recovery of the file system if the current owner fails. • Use the EXCLUDE list to prevent recovery of a file system from transferring ownership to a particular system, or set of systems, in the participating group. When the current owner fails, recovery of the file system is performed to an owner outside the exclude list. When the current owner fails, recovery of the file system is performed to a randomly selected owner outside the exclude list. • Use the INCLUDE list to ensure that recovery of a file system will transfer ownership only to a particular system or set of systems in the participating group. Recovery of the file system is performed in priority order only by the list of systems that are specified in the INCLUDE list. **Restriction:** Only use this option on mounts of file systems that are critical to operation across a subset of systems in the participating group, or when you do not want certain systems in the participating group to have ownership of the file system. If recovery processing fails to establish a new owner for the file system, the file system is unmounted, along with all the file systems mounted within it.

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 that are described in "Customizing BPXPRMxx for a shared file system" on page 165. See the associated documentation for the exact syntax.

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

In addition, when recovering from system outages, you need to weigh sysplex availability against availability to the server.

When an original owner system reinitializes, you might want to move the read/write file system back to this original owner if the original owner is the primary system that accesses the file system and if the PFS does not support accessing the read/write file system in a sysplex-aware mode. Better performance should occur when the file system is locally mounted (owned) at the system that most

frequently accesses the file system. See also “File system availability” on page 181 and “Tuning z/OS UNIX performance in a sysplex” on page 190.

Shared file system implications during a planned shutdown of z/OS UNIX

These sections contain the procedures to use when shutting down z/OS UNIX.

- “Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN” on page 262
- “Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=...” on page 258

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.

Use F OMVS,SHUTDOWN to shut down file systems. It is described in “What F OMVS,SHUTDOWN does” on page 261. If this is not appropriate for your installation, use the F BPXOINIT,SHUTDOWN=... procedure described in “Planned shutdowns using F BPXOINIT,SHUTDOWN=...” on page 258.

State of file systems after shutdown

File systems on the system where the shutdown was issued are immediately unmounted. As a result, data is synched to disk. For shared file systems, one of the following actions is done on the file systems that are owned by the system where the command was issued.

- Unmount if automounted or if a file system was mounted on an automounted file system.
- Move to another system if an AUTOMOVE was specified.
- Unmount for all other file systems.

File systems that are not owned by the system on which the shutdown was issued are not affected.

Initializing the file system

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 elsewhere in the sysplex. Although you can start multiple systems simultaneously, when they reach the point of z/OS UNIX initialization, their processing is serialized. Serialized processing allows only one system at a time to initialize z/OS UNIX.

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

- 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 successfully might become unavailable due to a system outage. When a system is removed from the sysplex, a window of time exists 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 were moved to other systems before the system exited. If that did not happen, a window of time exists 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 prevent 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.

Occasionally, dependencies might exist among the file systems that are being mounted by different systems in the participating group. If the systems are concurrently being initialized, the initial attempt to mount the file systems might fail in certain situations, such as when a mount point directory is not available because the containing file system has not yet been mounted by another system in the participating group. The failed mount requests are automatically reissued both during and after file system initialization. They might be reissued multiple times.

Locking files in the sysplex

You can lock all or part of a file that you are accessing for read/write purposes by using the byte range lock manager (BRLM).

The lock manager is initialized on every system in the sysplex. This is known as distributed BRLM, and it is the only supported byte range locking method. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system.

z/OS UNIX backs up each lock in the application's system when the actual lock is stored in another system. This redundant locking provides recovery of locks when a system in the sysplex terminates abnormally. If z/OS UNIX is able to successfully recover a file system, then it restores the locks that are held for files in that file system. Doing so prevents error conditions from occurring.

If the following situations occur, the locks for the corresponding files are considered to be lost:

- z/OS UNIX cannot recover the file system and unmounts it, including unmounting due to a MODIFY OMVS,STOPPFS command.
- NOAUTOMOVE was specified for the file system.
- Any lock for a file was not successfully backed up.

Existing applications that have locked those files are prevented from accessing those files. Specifically, after a failure where byte range locks are lost, z/OS UNIX provides the following information to processes that 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 that has used 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.

System completion codes in *z/OS MVS System Codes* describes the system completion code EC6 and its associated reason codes.

For more information about the pfsctl callable service, see pfsctl (BPX1PCT, BPX4PCT) — Physical file system control in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Note: The F BPXOINIT,SHUTDOWN=FILESYS|FILEOWNER and F OMVS,SHUTDOWN commands do not cause a signal to be generated just because an application has lost byte range locks, although a signal might be generated for other reasons during shutdown processing.

Mounting file systems by using symbolic links

You can mount different file systems at a logical mount point that resolves to a different path name on different systems.

While `$VERSION/` can be used to differentiate a path based on the version level of a system and `$SYSNAME/` can be used to differentiate on each system, you can use special identifiers to mount file systems that use symbolic links. These are `$SYSSYMR/template` and `$SYSSYMA/template`.

Restriction: When mounting file systems that use symbolic links, observe these restrictions:

1. Like `$VERSION/` and `$SYSNAME/`, the identifiers need to be at the beginning of the link name.
2. Only the first occurrence of `$SYSSYMR/` or `$SYSSYMA/` in the link name is recognized as an identifier for which the remaining text requires substitutions. Any other identifiers after the first one will remain as is in the resolved link name.
3. Text must follow a `$SYSSYMR/` or `$SYSSYMA/` in order for it to be recognized as a valid identifier with text containing symbols to be resolved.
4. Any system symbol in the symbolic link text that is recognized by the ASASYMBM service are resolved. However, only static system symbols should be used in order to avoid unexpected results. These symbols are assigned a value at initialization. For information about system symbols, see What are system symbols? in *z/OS MVS Initialization and Tuning Reference*.

You can use D SYMBOLS to display the current settings of system symbols.

These examples assume that the standard MVS symbol `&SYSR1.` resolves to `0SV315` on SY1 and resolves to `0SV315B` on SY2.

1. If the symbolic link is `/x/y/sym1`, and the symbolic link contains `$SYSSYMR/&SYSR1./resdir`, a path name lookup on `/x/y/sym1` from SY1 will resolve the symbolic link to `0SV315/resdir`. Because it is a relative path name (the identifier was `$SYSSYMR/`), the resulting path name will be `/x/y/0SV315/resdir`.

   For example, on a mount, passing `/x/y/sym1` as the input mount point path name, the mount point would be: `/x/y/0SV315/resdir` on SY1.

   - If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name would result in a mount point of `/x/y/0SV315B/resdir`.
   - On a v_readlink syscall, passing the VnToken for the symbolic link, the output link name would be `0SV315/resdir` on SY1 or `0SV315B/resdir` on SY2.
2. If the symbolic link is `/x/y/sym1`, and the symbolic link contains `$SYSSYMA/&SYSR1./resdir`, a path name lookup on `/x/y/sym1` from SY1 will resolve the symbolic link to `/0SV315/resdir`. Because it is an absolute path name (the identifier was `$SYSSYMA/`), the resulting path name will be `/0SV315/resdir`.

   For example, on a mount, passing `/x/y/sym1` as the input mount point path name, the mount point would be `/0SV315/resdir` on SY1.

- If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name from SY2 would result in a mount point of `/OSV315B/resdir`.
- On a v_readlink syscall, passing the VnToken for the symbolic link, the output link name would be `/OSV315/resdir` on SY1 and `/OSV315B/resdir` on SY2.

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 UNIX file hierarchy. From there, the user can create, delete, read, write, and treat the remotely located files as part of their 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. Entries in the NFS Server Export Data Set can control which UNIX directories can be mounted by client users. You can specify either fully qualified path names or symbolic links.

Tuning z/OS UNIX performance in a sysplex

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

For example, assume that a user on SY1 requests a read on a file system mounted read/write and owned by SY2. Using shared file system 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 about tuning the signal services in a sysplex environment, see Tuning the signaling services and coupling facility in *z/OS MVS Setting Up a Sysplex*.

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 many mounts are required on any system participating in a shared file system.

Chapter 8. Customizing the shells and utilities

This topic discusses how to customize the z/OS and tcsh shells and the common tasks that need to be done when setting up the utilities.

Lists of subtasks

Subtask	Associated procedure
Invoking the shell automatically under TSO/E	“Steps for enabling shell users to invoke the shell automatically ” on page 191
Customizing the `/etc/profile` file	“Steps for customizing /etc/profile” on page 198
Customizing the `/$HOME/.profile` file	“Steps for customizing $HOME/.profile” on page 200
Customizing the `/etc/rc` file	“Steps for customizing /etc/rc” on page 206
Customizing the `/etc/inittab` file	“Steps for customizing /etc/inittab” on page 208

Connecting to the shell

To work interactively, the shell user connects to the system in one of the following ways:

- Logs on to TSO/E and enters the TSO/E command OMVS, which invokes a shell. The OMVS command provides a 3270 terminal interface to the shell, and you can use the options to customize the interface - for example, function key settings.
- Issues the `rlogin` command, which invokes the shell. It provides an asynchronous terminal interface to the shell, which is familiar to UNIX users.
- Issues the `telnet` command, which invokes the shell. It provides an asynchronous terminal interface to the shell, which is familiar to UNIX users.

After the user logs in to the shell, the system initializes the shell. See “Customizing the z/OS UNIX shells ” on page 193 for information about customizing the shell invocation.

Invoking the shell automatically under TSO/E

A shell user can invoke the shell automatically at the end of logging on to TSO/E. For the automatic invocations, the system invokes the shell after the TSO/E logon completes initialization. Automatic invocation is a handy way to enter the same OMVS command with options each time.

The automatic invocation can be set up by the system programmer or by the shell user.

Steps for enabling shell users to invoke the shell automatically

Before you begin, you need to know which shell users want to invoke the shell automatically when they log on to TSO/E, and you must be the system programmer.

Perform the following steps to enable shell users to invoke the shell automatically when they log on to TSO/E.

1. Select or create a TSO/E logon procedure for users who want to invoke the shell automatically when they log on to TSO/E

---

2. In the logon procedure, add a PARM parameter to the EXEC statement for program IKJEFT01.

```
//      EXEC  PGM=IKJEFT01, ...
//            PARM=OMVS
```

_____

3. Tell users to specify the procedure on the TSO/E logon panel on the line:

```
Procedure ===>
```

_____

When you are done, you have enabled shell users to invoke the shell automatically when they log on to TSO/E.

Tip:

- To customize the OMVS command for all shell users, you can create a REXX exec with the customized options. Then specify the name of the REXX exec in the PARM parameter, instead of with the OMVS command. In the exec, for example, you can specify the following changes:
  - Use of the 3270 alarm.
  - Number of sessions. (The default is 1.) For example:

    ```
    OMVS SESSIONS (3)
    ```

  - The key or keys to be used for escape.
  - The settings for the function keys.
  - The table to be used for code page conversion.
  - Shared address space.
- To customize any of the default function key settings, type your selection within the parentheses after the function key name.

  To make function key 1 (<PF1>) the Control key, issue:

  ```
  OMVS PF1(CONTROL)
  ```

  Use it to type an escape sequence such as <Ctrl-D>. (First type d on the command line and then press the function key.)

You can now perform the steps for the decision you have made.

Invoking the shell automatically when logging on to TSO/E

A TSO/E user can invoke the shell automatically when logging on to TSO/E. When logging on, you can invoke the shell by adding one of the following to the TSO/E logon panel:

- The OMVS command.

  For example, if the default options are required:

  ```
  Command ===> OMVS
  ```

  Or:
- The name of the REXX exec that contains an OMVS command with operands for required options.

  For example, if the exec name is MYOM:

  ```
  Command ===> MYOM
  ```

TSO/E processes this command each time the user logs on until the user deletes the command from the panel or changes it.

You might want to invoke OMVS from ISPF for the following reasons:

1. It is faster to invoke an ISPF dialog such as OEDIT or OBROWSE because ISPF does not need to start and stop to run the command.
2. It is not necessary to type *** and press **<Enter>** after running an ISPF dialog; control returns to the shell.
3. You can use split-screen support when using an ISPF dialog.

For information about REXX, see *z/OS TSO/E REXX Reference* and *z/OS TSO/E REXX User's Guide*

Determining the CPU time limit

The time limit for using a shell is the same as the TSO/E timeout. In determining the time, the system does not count the processing time for shells running in separate address spaces or processes forked by the shell. If you specify environment variable _BPX_SHAREAS=YES, then the shell processes and possibly one shell command are created in local processes. The CPU time consumed by local processes comes out of the TSO/E address space's time limit.

Supplying an alternative shell

If your installation decides to supply its own shell, consider doing the following tasks:

- Install the alternative shell in the file system.
- Set the sticky bit on and put the alternative shell in the link pack area (LPA).
- Specify the path name of the shell in the PROGRAM parameter for the OMVS RACF profile for users who want to use this as their default shell.
- Customize `/etc/init.options` if the shell script used for system initialization will use this shell.
- Identify the alternative shell in an input parameter in the `/etc/init.options` file.

If you are using the `/etc/profile` for the z/OS shell or `/etc/csh.login` for the tcsh shell, you might need to review them and make any necessary adjustments.

For an example of MAXPROCSYS settings in BPXPRMxx, see "Monitoring use of system resources" on page 362.

Customizing the z/OS UNIX shells

After a user logs in to the shell, the system initializes the shell for that user. During the initialization, the system does the following:

1. Determines whether the user is authorized to use the shell by checking for a UID value in the user's RACF user profile. It also checks that the user's RACF group has a GID assigned to it.
2. Sets the LOGNAME, HOME, and SHELL environment variables from data in the RACF user profile, which was specified in the RACF ADDUSER and ALTUSER commands. See "Customizing the RACF user profile" on page 194.
3. Connects the user to the initial working directory that was identified in the HOME environment variable in the RACF user profile. If the RACF user profile does not identify a working directory, the system uses the root as the user's working directory and issues a message.
4. Invokes the shell named in the PROGRAM statement of the OMVS segment in the RACF user profile.
   - For the z/OS shell, it is named `/bin/sh`.
   - For the tcsh shell, it is named `/bin/tcsh`.

Similar systems typically have an `/etc/passwd` file, which contains the HOME and PROGRAM environment variables. The file also contains the passwords and password phrases that are used. To provide better security, the z/OS shell does not use the `/etc/passwd` file; instead, it uses the initial values assigned to these variables in the RACF user profiles. RACF maintains the passwords and password phrases.

Customizing the shell environment variables

If an environment variable is not set, it has no value. Setting an environment variable is optional.

For the z/OS shell

The places to set environment variables, in the order that the system sets them, are:

1. The RACF user profile.
2. The `/etc/profile` file, which is a system-wide file that sets environment variables for all z/OS shell users. This file is only run for login shells.
3. The `$HOME/.profile` file, which sets environment variables for individual users. This file is only run for login shells.
4. The file named in the ENV environment variable. This file is run for both login shells and subshells.
5. A shell command or shell script.

Later settings take precedence. For example, the values set in `$HOME/.profile` override those in `/etc/profile`.

Depending on the commands used to set it, an environment variable can be local (used only for the current process) or exported (used for the current process and for any other processes spawned by the current process).

For the tcsh shell

The places to set environment variables, in the order that the system sets them, are:

1. The RACF user profile
2. The `/etc/csh.login` file, which is the system-wide file that sets environment variables. This file is only run for login shells.
3. The `$HOME/.login` file, which sets environment variables for individual users. This file is only run for login shells.
4. The `/etc/csh.cshrc` file, which is the system-wide file that sets shell variables, some environment variables (like PATH), and umask. It also defines command aliases. It is used by subshells.
5. The `$HOME/.tcshrc` file, which sets environment variables for individual users. It is used by subshells.
6. The `$HOME/.cshrc` file, if it is provided for compatibility with the C shell.

Later settings take precedence. For example, the values set in `$HOME/.login` override those in `/etc/csh.login`.

Customizing the RACF user profile

The security administrator defines a user by creating a RACF user profile with an ADDUSER command or alters the user profile with an ALTUSER command. The RACF user profile contains values that are used to set the following environment variables:

LOGNAME
The TSO/E user ID.

HOME
The path name of the user's home directory as specified in the HOME parameter of the RACF command. If the HOME parameter was not specified, HOME is the root directory. Unpredictable results might occur if an invalid or nonexistent directory is specified for the HOME parameter. For more information about setting up home directories for users, see "Defining z/OS UNIX users to RACF" on page 51.

SHELL
The path name of the file containing the shell program as specified in the PROGRAM parameter on the RACF command. If PROGRAM was not specified, SHELL is `/bin/sh`.

The PROGRAM parameter can specify a special-purpose shell or another kind of program.

Customizing files for the z/OS shell

You might want to customize these system-wide profiles that you might want to customize to meet the needs of your installation:

- `/etc/profile`
- `$HOME/.profile`
- `/etc/init.options`
- `/etc/rc`
- `/etc/inittab`

Customizing /etc/profile

The `/etc/profile` file is the system-wide profile for the z/OS shell users. It contains environment variables and commands that are used by most shell users. To improve system performance, use STEPLIB=none.

Note: Because `/etc/profile` is the z/OS shell equivalent to `/etc/csh.login` for tcsh, you must keep system-wide information for both sets of users in sync. Any customization for `/etc/profile` (such as setting environment variables) must be duplicated in C-shell syntax in `/etc/profile`. Future changes to `/etc/profile` also must be made to `/etc/csh.login`. If you maintain a non-z/OS UNIX system, you could consider porting `/etc/profile` and `/etc/csh.login` from that system to the z/OS system and merging them with the z/OS samples.

Copy the IBM-supplied `/samples/profile` file to `/etc/profile`. (See "Steps for customizing /etc/profile" on page 198 for the procedure.) The following extract shows an excerpt of `/samples/profile`.

```
# ====================================================================
#                    STEPLIB environment variable
#                    ----------------------------
# Specifies a list of data sets to be searched ahead of the normal
# search order when executing a program. To improve the shell's
# performance for users from ISPF or users with data sets allocated to
# STEPLIB DD statements, specify "STEPLIB=none" .
# This performance improvement is not applicable to non-interactive
# shells, for example those started with the BPXBATCH and OSHELL
# utilities.
# The exec will restart the current shell, as a login shell with the
# same argv[0] name.
# ====================================================================

if [ -z "$STEPLIB" ] && tty -s;
then
       export STEPLIB=none
       exec -a $0 $SHELL
fi

# ====================================================================
#                      TZ environment variable
#                      -----------------------
# Specifies the local time zone.
# ====================================================================
TZ=EST5EDT
export TZ
```

```
# ====================================================================
#                     LANG environment variable
#                     -------------------------
# Specifies the language you want the messages to displayed in.
# For Japanese: LANG=Ja_JP
# ====================================================================
LANG=C
export LANG

# ====================================================================
#                   LOGNAME environment variable
#                   ----------------------------
```

```
# This environment variable is set when 'logging' into the shell
# environment. You can avoid accidental modification to this variable
# by making the LOGNAME variable read-only.
# ================================================================
readonly LOGNAME

# ================================================================
#                     PATH environment variable
#                     -------------------------
# Specifies the list of directories that the system searches for an
# executable command. If you want to include the current working
# directory in your search order, then the enviroment variable would
# be
#   PATH=/bin:.

#
# The current working directory is represented by dot ('.') .
# ================================================================
PATH=/bin
export PATH

# ================================================================
#                    LIBPATH environment variable
#                    ----------------------------
# Specifies the list of directories that the system searches for a DLL
# (Dynamic Link Library) filename. If not set, the current working
# directory is searched.

LIBPATH=/lib:/usr/lib:.
export LIBPATH

# ================================================================
##                    NLSPATH environment variable
#                     ----------------------------
# Specifies the list of directories that the system searches for
# message catalogs (NLS files). The %L represents the language currently
# set by the LANG environment variable, and %N represents the name
# of the message catalog.
# ================================================================

NLSPATH=/usr/lib/nls/msg/%L/%N
export NLSPATH

# ================================================================
#                    MANPATH environment variable
#                    ----------------------------
# Specifies the list of directories that the system searches for man
# pages (help files). The %L represents the language currently set by
# the LANG environment variable.
# ================================================================
MANPATH=/usr/man/%L
export MANPATH

# ================================================================
#                     MAIL environment variable
#                     -------------------------
# Sets the name of the user's mailbox file and enables mail
# notification.
# ================================================================
MAIL=/usr/mail/$LOGNAME
export MAIL

# ================================================================
#                           umask variable
#                           --------------
# Sets the default file creation mask - reference umask in the z/OS
# UNIX System Services Command Reference
# ================================================================
umask 022

# ================================================================
# Start of c89/cc/c++ customization section
# ================================================================
#
#   The following environment variables are used to provide information
#   to the c89/cc/c++ utilities, such as (parts of) data set names which
#   are dynamically allocated.
#
# ################################################################################
#
# for _CMP in _C89 _CC _CXX; do
##
```

```
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# ============================================================================
#
##  C/C++ Compiler:
#   ---------------------------------------
#   export ${_CMP}_CLIB_PREFIX="CBC"
##
#   Prelinker and runtime library:
#   ---------------------------------------
#   export ${_CMP}_PLIB_PREFIX="CEE"
##
#   z/OS system data sets:
#   ---------------------------------------
#   export ${_CMP}_SLIB_PREFIX="SYS1"
#
#
# Compile and link-edit search paths:
# ============================================================================
##
#   Compiler include file directories:
#   ---------------------------------------
#   export ${_CMP}_INCDIRS="/usr/include /usr/lpp/cbclib/include"
##
#   Link-edit archive library directories:
#   ---------------------------------------
#   export ${_CMP}_LIBDIRS="/lib /usr/lib"
##
# Esoteric unit for data sets:
# ============================================================================
##
#   Unit for (unnamed) work data sets:
#   ---------------------------------------
#   export ${_CMP}_WORK_UNIT="SYSDA"
##
# done; unset _CMP
#
# ============================================================================
# End of c89/cc/c++ customization section
```

Some of the statements in `/etc/profile` are explained in the following list:

STEPLIB=none
Indicates that STEPLIBs should be not propagated. Running the shell with STEPLIB=none assumes that the Language Environment runtime library resides in LINKLIST or in LPA. It is run only on the first invocation of an interactive shell.

exec -a $0 $SHELL
Restarts the current shell in the current address space with the environment variables that you have just defined. The "if test" causes this to be run only on the first invocation of an interactive shell. The `tty -s` test prevents the shell from being run by noninteractive invocations such as those started with BPXBATCH and OSHELL.

TZ=EST5EDT
Sets the time zone. In the sample profile, TZ is set to EST5EDT, which is Eastern Daylight Time.

LANG=C
Specifies the name of the default locale. `C` specifies the POSIX locale and `Ja_JP` specifies the Japanese locale.

readonly LOGNAME
Sets the LOGNAME variable to read-only so that it is not accidentally modified.

PATH=/bin
Sets a default command search path to search only the `/bin` directory.

LIBPATH=/lib:/usr/lib:.
Specifies the directory to search for a dynamic link library (DLL) file name. If LIBPATH is not set, only the working directory is searched.

NLSPATH=/usr/lib/nls/msg/%L/%N
Sets the path for message catalogs.

MANPATH=/usr/man/%L
Sets the path for the man pages.

When you set the MANPATH environment variable, you are telling the shell where to find the man pages. The value of the LANG environment variable is substituted for %L in this directory and in the directories that are specified by MANPATH. The default man page is located in the /usr/man/C directory. If the value of LANG environment is set to a non-C value, you must make sure that man pages exist in the /usr/man/%L directory according to the value of LANG environment.

MAIL=/usr/mail/$LOGNAME
Sets the name of the system mail file and enables mail notification.

umask 022
Sets the default file creation mask. In the sample, the mask is set to 022. This causes a file that is created with mode 777 to have permissions of 755. The creator cannot set the group write or other write bits on in the file mode field because the mask sets them off.

export TZ PATH NLSPATH MAIL LANG
Exports the values so the system will have access to them.

Steps for customizing /etc/profile

Before you begin: You need to have access to /samples/profile.

Perform the following steps to customize /etc/profile.

1. Copy /samples/profile to /etc/profile. If you already have /etc/profile, then compare it to /samples/profile and retrofit any new updates. For example:

   ```
   cp /samples/profile  /etc/profile
   ```

   _____

2. Edit v to change and add environment variables. For more information about environment variables that you can use, see “Environment variables that you can customize for /etc/profile” on page 198.

   _____

When you are done, you have customized /etc/profile.

Environment variables that you can customize for /etc/profile

TZ
Identifies the time zone used by most of your users. It gives the standard time zone, the number of hours offset from Coordinated Universal Time Coordinated (UTC) – also called Greenwich Mean Time (GMT) – and the daylight saving time zone.

1. For a system whose users are mostly in New York and Boston, this variable would be:

   ```
   TZ=EST5EDT
   ```

2. For a system with most users in Houston, this variable would be:

   ```
   TZ=CST6CDT
   ```

PATH
Defines the default command path. This variable should name all directories in which the installation plans to put utilities and licensed programs.

If you plan to place all standard utilities in the /bin directory, this variable is:

```
PATH=/bin
```

If you want your users to access another product's binaries that were installed into /usr/lpp/xxxxxxxx/bin, the variable becomes:

```
PATH=/bin:/usr/lpp/xxxxxxxx/bin
```

The order of the directories in the PATH environment variable controls the search order.

To add the working directory to the search, add a colon and a period (`:.`) as follows:

```
PATH=/bin:/usr/lpp/xxxxxxxx/bin:.
```

To search the working directory first, specify:

```
PATH=.:/bin:/usr/lpp/xxxxxxxx/bin
```

FPATH
Contains a list of directories that the z/OS shell searches to find shell functions. The `/samples/profile` file does not have a default FPATH setting. Add the definition of the FPATH environment variable to point to the directories in `/etc/profile` that contain the shell function definitions.

Directories in this list are separated by colons. Every directory is searched in the order specified in the list until a matching function definition file is found. If you have shell functions that you want to make available to all users, do the following:

1. Define a directory that is readable by all users.
2. Put the shell function definitions in files within this directory.
3. Add the definition of the FPATH environment variable to `/etc/profile`. Mark it as an exported variable with the `export` command.

Example: A function named `buildapp` is in a file named `/usr/lib/appdev/functions/buildapp`. Add the following statement to `/etc/profile`:

```
export FPATH=/usr/lib/appdev/functions
```

The user can then just issue `buildapp`. The first time `buildapp` is run, it is found in FPATH, defined in the current shell, and executed. After that first time, every time `buildapp` is issued (within the same shell), the shell executes `buildapp` without first searching for that function.

FPATH follows the same format as the PATH environment variable.

NLSPATH
Specifies the path that the messaging service will use to find a message catalog.

LANG
Contains the default locale name.

MAIL
Define the name of the system mail file and enables notification of mail. If you plan to use a mail file other than `/usr/mail` (for example, `/usr/notes`), set the variable as follows:

```
MAIL=/usr/notes/$LOGNAME
```

STEPLIB
Defines libraries that should be searched to load MVS load modules. Normally, installations should specify STEPLIB=NONE to prevent the propagation of STEPLIBs. If a STEPLIB environment variable is needed, specify only the required library. For example:

```
STEPLIB=CEE.SCEERUN:CEE.SCEERUN2
```

If you do not specify the STEPLIB environment variable, STEPLIBs are propagated from the user's TSO/E user ID. Specifying a value other than STEPLIB=NONE can affect performance for the following reasons:

- Each time a fork or exec is invoked, STEPLIB data sets are dynamically allocated for the user.
- Each time an MVS load module is loaded, the STEPLIB data set directories are searched.
- Each time an MVS load module is found in the STEPLIB concatenation, the module is loaded from there into the user's private area storage.

LOCPATH
Tells the setlocale() function the name of the directory from which to load locale object files. If LOCPATH is not defined, the default directory `/usr/lib/nls/locale` is searched.

LOCPATH is similar to the PATH environment variable. It contains a list of z/OS UNIX directories that are separated by colons.

For information about how setlocale() searches for locale object files, see setlocale() in *z/OS XL C/C++ Runtime Library Reference*.

Customizing $HOME/.profile

The optional `$HOME/.profile` file contains commands that set or change the values of environment variables for an individual user. (HOME is a variable for the path name for a user's home directory.) The values set in `$HOME/.profile` can override those in `/etc/profile`.

Because `$HOME/.profile` is the z/OS shell equivalent to `$HOME/.login` for tcsh, you must keep system-wide information for both sets of users in synch. Any customization that you did for `$HOME/.profile` (such as setting environment variables) must be duplicated in C-shell syntax in $HOME/.login. Future changes to `$HOME/.profile` also must be made to `$HOME/.login`. If you maintain a non-z/OS UNIX system, you could consider porting `$HOME/.profile` and `$HOME/.login` from that system to the z/OS system and merging them with the z/OS samples.

The following example shows the IBM-supplied `/samples/.profile`.

```
# ENV=$HOME/.setup
# export ENV

PATH=$PATH:$HOME:

EDITOR=ed

PS1='$LOGNAME':'$PWD':' >'

export PATH EDITOR PS1
```

Steps for customizing $HOME/.profile

Before you begin: You need to have access to `/samples/.profile`.

Perform the following steps to customize `$HOME/profile`.

1. Copy `/samples/.profile` to your $HOME directory. For example:

   ```
   cp /samples/.profile  $HOME/.profile
   ```

   _____

2. Edit `.profile` to change and add environment variables. For more information about environment variables that you can use, see "Environment variables that you can customize for $HOME/.profile" on page 200.

   _____

When you are done, you have customized `$HOME/profile`.

Environment variables that you can customize for $HOME/.profile

Table 29 on page 201 lists the environment variables that you can customize for `$HOME/.profile`.

Table 29. Environment variables that you can customize for $HOME/.profile	
Environment variable	**What it does**
ENV	Specifies the name of the user's environment file, which is a shell script. ENV=$HOME/`.setup` specifies a file called `.setup`, which the user added to the home directory.
STEPLIB	Specifies STEPLIBs for individual users who have STEPLIB requirements that are different from those of other users. **Tip:** Use STEPLIB=none. However, there might be cases in which a specific library is needed; for example, STEPLIB=USER1.MY.USERLIB.
PATH	Appends your home directory to the current path. When you set up your own $HOME/`.profile` as superuser, specify the `/usr/sbin` directory in your PATH variable because some superuser utilities are in that directory instead of the `/bin` directory. Those utilities include **`automount`**, **`inetd`**, **`mknod`**, **`rlogind`**, **`uucpd`**, **`chroot`** and **`cron`**
PS1	Specifies the prompt character or string.
TZ	Specifies a different time zone if the user is in a remote location.
MAILRC	Specifies the name of the user's mail startup file. The default is $HOME/`.mailrc`.
MAIL	Specifies the name of the user's file for mail that the user does not save in some other file. The default is $HOME/`mbox`.
MBOX	Specifies the name of the user's file for mail that the user does not save in some other file. The default is $HOME/`mbox`.
DEAD	Specifies the name of the user's file for partial messages when an interrupt or error occurs when creating a message or delivering it. The default is $HOME/`dead.letter`.

In order for system programmers and administrators to run authorized utilities and to start daemons found in `/usr/sbin`, they must have a $HOME/`.profile` file with the PATH environment variable set as follows:

```
PATH=$PATH:/usr/sbin:$HOME:
```

Using an ENV environment variable file

When the shell is invoked as a login shell, `/etc/profile` and $HOME/`.profile` are used. If you want environment variables in a shell invoked from the current shell, you should identify a file in an ENV environment variable and place shell commands in the file. The shell interprets the file named in ENV each time a new shell is invoked.

Using a shell command or shell script

For the current shell invocation, a user can enter a shell command to set the value of any environment variable. Any variables set in a shell script are set only while the script is running and do not affect the shell that invoked the shell script (unless the script is “sourced” by running it with the dot command).

Customizing /etc/init.options

The file /etc/init and /usr/sbin/init are referred to synonymously as the initialization program that is run when the OMVS address space is initialized.

The /usr/sbin/init program invokes a shell to execute an initialization shell script that customizes the environment. When this shell script finishes or when a time interval established by /usr/sbin/init expires, kernel services become available for general batch and interactive use. Standard output (stdout) and standard error output (stderr) are redirected to /etc/log.

Table 30 on page 202 lists the files that are associated with /usr/sbin/init.

Table 30. Files that are associated with /usr/sbin/init

File	What it is
/bin/sh	The default shell that /usr/sbin/init invokes to execute /etc/rc or another shell script that is specified in the /etc/init.options file.
/etc/init.options	The customized initialization options file, which is read by /usr/sbin/init.
/etc/rc	The default shell script that is used for initialization.
/etc/log	The file that output is written to.
Other utilities	Services that are called by the initialization shell script.

/usr/sbin/init and the customized /etc/init.options and /etc/rc are run at IPL. There is no other way to invoke them explicitly.

Before /usr/sbin/init invokes the shell to execute the system initialization shell script, it reads the file /etc/init.options for values of various options. The IBM-supplied default is in /samples/init.options. Copy this file to /etc/init.options and make the appropriate changes. If you already have /etc/init.options, then compare it to /samples/init.options and retrofit any new updates.

/usr/sbin/init treats all lines in /etc/init.options that do not start with a hyphen (-) as comment lines. Lines that start with a hyphen are used to specify options. The format of lines specifying options is as follows:

```
-oo  vvvvv comment
```

where:

- oo is a field of one or more nonblank characters immediately following the hyphen that identifies the option. The end of the option field is delimited by one or more blanks.
- vvvvv is a field of one or more nonblank characters that specify an option value. These characters are numeric, alphabetic, or a combination of both, depending on the option being specified. The end of the value field is delimited by one or more blanks.

  Option and option value characters must appear in columns 1 through 79 of an option line in /etc/init.options. /usr/sbin/init ignores characters beyond column 79. However, a backslash (\) immediately following nonblank value field characters is recognized as a continuation character. If the continuation character is found, nonblank characters at the beginning of the next line are treated as option value characters. The first blank character delimits the end of the value field.

  Option value characters on a continuation line are limited to columns 1 through 79.

  The continuation character is recognized on continuation lines as well as the option line.
- Any characters after a blank delimiting the end of the option value field on the same line are treated as comment characters.

Options and option value ranges are listed as follows:

-a ***nnnn***
Alarm option. Specifies the maximum time in seconds allowed for the shell script to complete. You must specify enough time for the system initialization script to complete if this is a requirement at your installation.

The default is 180 seconds.

The maximum is 9999 seconds.

If the shell does not signal completion of the script before this time elapses, `/usr/sbin/init` writes the timeout error message, FSUM4013I, in `/etc/log` and exits.

If the value `0` is specified, no timeout interval is set. The decision to specify the value `0` for the alarm option should be made carefully and only after you know that the initialization script is error-free.

-t ***n***
Terminate option. Specifies whether to end the shell script initialization if the timeout specified by the alarm option (`-a`) occurs.

0
Allows the shell script to continue

>0
Ends the shell script

1
The default; ends the shell script.

If you specify terminate and the timeout waiting for the initialization shell script occurs, `/usr/sbin/init` sends a stop signal to the shell process group.

It is your responsibility to decide if the initialization shell script can continue concurrently with batch and interactive use of the shell.

-e ***string***
Environment variable option: *string* in the form *name=value* specifies the environment variable name and the value that `/usr/sbin/init` passes to the shell that it is invoking

The maximum length is 255 characters.

`/etc/init.options` can contain up to 25 `-e` option lines specifying names and values for different environment variables. `/usr/sbin/init` passes the resultant environment variable array to the shell that it invokes. In turn, the shell uses this array to set up an execution environment for the initialization shell script that is appropriate for the installation. TZ is an example of an environment variable that should be considered.

These environment variables should also be set up in `/etc/profile` or `$HOME/.profile` for each interactive user. Examples of variables that you could specify are TZ, LANG, and NLSPATH.

-sc ***pathname***
The path name of the initialization shell script.

The default is `/etc/rc`.

The maximum length is 255 characters.

-sh ***pathname***
pathname specifies the shell to be invoked by `/usr/sbin/init` to run the initialization script. `/usr/sbin/init` cannot set environment variables for the rest of the system.

The default is `/bin/sh`.

The maximum length is 255 characters.

`-sh` <blanks> tells `/usr/sbin/init` not to run the shell. Instead, `/usr/sbin/init` signals that multiuser mode is to be entered and then exits.

Following is a sample /etc/init.options file showing the time zone, the Japanese language, and the locale:

```
-e TZ=JST-9
-e LANG=Ja_JP
-e NLSPATH=/usr/lib/nls/msg/%L/%N
```

/etc/init opens the message catalog fsumucat.cat in directory /usr/lib/nls/msg/C unless an NLSPATH environment variable naming a different directory is specified in the /etc/init.options file.

For more information about national language support, see Chapter 9, “Customizing for your national code page in the shell,” on page 219.

Tip: You can use a REXX exec in an MVS data set as an alternative to running the /etc/init initialization program. To activate the REXX exec for initialization, you must specify its name on the STARTUP_EXEC statement in the BPXPRMxx parmlib member.

Customizing /etc/rc

The /etc/rc file contains customization commands for z/OS UNIX System Services Application Services. Copy the /samples/rc file that is supplied from IBM to /etc/rc, as described in “Steps for customizing /etc/rc” on page 206 and customize it.

If the /etc/inittab file is installed, z/OS UNIX installation processes the /etc/inittab file instead of executing the /etc/rc file. If you want to use /etc/inittab in addition to the existing /etc/rc file, then include an entry in the /etc/inittab file to start /etc/rc.

The following excerpt of the /samples/rc file that is supplied from IBM includes the set -v -x command, which specifies that a verbose shell command trace of /etc/rc is to be written to /etc/log. Certain comments are also commented out.

```
# Initialization shell script, pathname = /etc/rc
#
#    LICENSED MATERIALS - PROPERTY OF IBM
#    5655-ZOS COPYRIGHT IBM CORP. 1993, 2023
#

# Initial setup for z/OS UNIX
export _BPX_JOBNAME='ETCRC'

# Provide z/OS UNIX Startup Diagnostics
set -v -x

# Setup utmpx file
>/etc/utmpx
chmod 644 /etc/utmpx

# Reset all subsidiary tty files
chmod 666 /dev/tty*
chown 0 /dev/tty*

# Allow only file owner to remove files from /tmp
chmod 1777 /tmp
# Allow only file owner to remove files from /var
chmod 1777 /var
# Allow only file owner to remove files from /dev
chmod 1755 /dev
# Allow only file owner to remove files from /var/spool/uucppublic
chmod 1777 /var/spool/uucppublic

# Setup write, talk, mesg utilities
# chgrp TTY   /bin/write
# chgrp TTY   /bin/mesg
# chgrp TTY   /bin/talk
# chmod 2755  /bin/write
# chmod 2755  /bin/mesg
# chmod 2755  /bin/talk
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO

# Setup mailx utility
# No need to CHGRP /usr/mail directory
# No need to CHGRP mailx utility
```

```
# No need to CHMOD mailx to turn on SETGID

# Setup uucp utility
# chown uucp:uucpg /usr/lib/uucp
# chown uucp:uucpg /usr/lib/uucp/IBM
# chown uucp:uucpg /usr/spool/uucp
# chown uucp:uucpg /usr/spool/locks
# chown uucp:uucpg /usr/spool/uucppublic
# chown uucp:uucpg /usr/spool/uucp/.Xqtdir
# chown uucp:uucpg /usr/spool/uucp/.Sequence
# chown uucp:uucpg /usr/spool/uucp/.Status
# chown uucp:uucpg /bin/uucp
# chown uucp:uucpg /bin/uuname
# chown uucp:uucpg /bin/uustat
# chown uucp:uucpg /bin/uux
# chown uucp:uucpg /usr/lib/uucp/uucico
# chown uucp:uucpg /usr/lib/uucp/uuxqt
# chown uucp:uucpg /usr/lib/uucp/uucc
# chmod 4755 /bin/uucp
# chmod 4755 /bin/uuname
# chmod 4755 /bin/uustat
# chmod 4755 /bin/uux
# chmod 4754 /usr/lib/uucp/uucico
# chmod 4754 /usr/lib/uucp/uuxqt
# chmod 4754 /usr/lib/uucp/uucc
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO

# Invoke vi recovery
#
#
# mkdir -m 777 /var/tmp
# export TMP_VI="/var/tmp"
if [ -e /etc/recover ]
then
    chmod 1777 /etc/recover
else
    mkdir -m 1777 /etc/recover
fi
/usr/lib/exrecover

# Create TERMINFO database
# tic /usr/share/lib/terminfo/ibm.ti
# tic /usr/share/lib/terminfo/dec.ti
# tic /usr/share/lib/terminfo/wyse.ti
# tic /usr/share/lib/terminfo/dtterm.ti
# commented tic out in HOT1180 - all TERMINFO files are shipped

#--------------------------------------------------------------------------------
# Added in HOT7750 to eliminate a manual migration action.
#
# Remove all files in the man cache that were created by the man command for
# the default MANPATH directory.
#
# NOTE: This loop only removes files for a subset of the LANG values
#       possible.  Administrators should customize the loop to include
#       the LANG values supported on their system.
#--------------------------------------------------------------------------------
for MAN_CACHE_LANG in C Ja_JP Zh_CN ; do
  [[ -d /var/man/$MAN_CACHE_LANG ]] && rm /var/man/$MAN_CACHE_LANG/*.[0-9].*
done

# Start the INET daemon for remote login activity
#_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &

sleep 5
echo /etc/rc script executed, `date`
```

The following setup sections have been commented out:

- For the **mailx** utility because the **mailx** utility no longer requires these entries.
- For creating the terminfo database because IBM ships the individual files that make up the terminfo database.
- For the **mesg**, **talk**, **write**, and **uucp** because this customization is now done when the FOMISCHO sample job is run.

Steps for customizing /etc/rc

Before you begin, you need to have access to /samples/rc. If you already have /etc/rc, then compare it to /samples/rc and retrofit any new updates.

Perform the following steps to customize /etc/rc.

1. Copy /samples/rc to the /etc/rc file. For example:

   ```
   cp /samples/rc  etc/rc
   ```

   _____

2. Edit /etc/profile to add additional entries, such as those for shell commands. For instance, you could add a command to start an installation-supplied daemon. The script can also invoke other scripts such as an rc.tcpip script to start tcp daemons.

   _____

3. Save the file.

   _____

When you are done, you have customized /etc/rc.

Customizing /etc/inittab

The /etc/inittab file lists system processes (for example, commands and shell scripts) that are started when z/OS UNIX is initialized. The file also identifies processes that can be restarted automatically when they end.

To create the/etc/inittab file, copy the /samples/inittab file to /etc/inittab as described in "Steps for customizing /etc/inittab" on page 208 and add additional entries to it. For example, if you want to take advantage of the respawn capability, you can add an inittab entry to start daemons such as syslogd and cron from /etc/inittab instead of from /etc/rc. When the respawn capability is used, programs that end are automatically restarted or respawned.

Tip: When the /etc/inittab file is installed, it is processed instead of the /etc/rcfile. Do not create the /etc/inittab file if you want the /etc/rc file to be run directly from the initialization process. If you want to use /etc/inittab in addition to the existing /etc/rc file, then include an entry in the /etc/inittab file to start /etc/rc. For example:

```
etcrc::wait:/etc/rc >/etc/log 2>&1
```

Rule: The /etc/inittab file must be installed into the file system before the system is IPLed or before it is restarted. It is processed only once during initialization and also once during OMVS restart.

Format of the /etc/inittab file

The /etc/inittab file is composed of entries that are position-dependent and have the following format:

```
Identifier:RunLevel:Action:Command
```

The colon character (:) is used as a delimiter. To comment out an entry in the /etc/inittab file, add : or # at the beginning of the entry.

For example, the following entry is commented out.

```
:Identifer:RunLevel:Action:Command
```

Each entry is delimited by a newline character. A backslash (\) character that precedes a newline character indicates the continuation of an entry. There are no limits on the number of entries in the /etc/inittab file. The maximum entry size is 1024 characters. The entry fields are:

Identifier
A string of 1 to 7 characters that uniquely identifies the entry. The ID is used as the job name for the process.

- The identifier must start in column 1 of the file.
- Valid characters are A-Z and 0-9. Lowercase characters are converted to uppercase.
- The identifier must start with an alphabetic character.

The identifier is required; there is no default.

RunLevel
Not supported on z/OS UNIX. Identifies the run levels in which this entry can be processed. Even though the RunLevel field is not supported, it is in the `inittab` entry for compatibility with other UNIX implementations. The run level field can be up to 32 alphanumeric characters. If it is specified, it will be checked for validity and the `inittab` entry will be skipped if it is in error.

Action
Specifies how to handle the process that is started in the command field. The supported actions are:

once
Starts the process and does not wait for it to end. Continues scanning the `/etc/inittab` file and processes the next entry. When it ends, the process is not restarted.

respawn
Starts the process and does not wait for it to end. Continues scanning the `/etc/inittab` file and processes the next entry. The process is restarted when it ends. When a process is spawned again, it is restarted with the same file descriptors and environment variables that it was started with originally.

If a process ends due to a shutdown of all fork activity, the process is not restarted until fork activity is re-enabled. If a respawnable process ends and then ends again after being restarted within 15 minutes of its first ending, then message BPXI082D is displayed. The message identifies the process entry and asks whether to try again or ignore the error. A process identified for respawn will not be able to register as a permanent process that can survive a shutdown and restart cycle because the `/etc/inittab` file will be processed again during restart.

Restriction: Daemons, such as `cron`, `inetd`, and `sshd` (the OpenSSH daemon), cannot be restarted using the respawn attribute. These processes fork themselves. The respawn attribute is associated with the parent process that is started, not the forked child processes.

To check the status of the respawn attribute, issue the D OMVS operator command and check the STATE field. You can also use the `-o attr` option of the `ps` shell command, which displays the process attributes.

To avoid excessive consumption of common storage, limit the number of processes started with the respawn attribute to 100 or fewer.

respfrk
Starts the process and does not wait for it to end. Continues scanning the `/etc/inittab` file and processes the next entry.

- If the process never issues a fork command, then this action behaves the same way as the respawn action.
- If the process issues a fork, then the respawn attribute is transferred to the forked child process. When the child process ends, the original parent process is spawned. This process will fork itself, thus restarting the daemon process.
- If the original process issues any additional forks, the respawn attribute is not transferred to those forked children. It is only transferred the first time the fork is issued.

The `respfrk` option is intended for a program that forks itself to create a child process which then continues running while the parent process ends. For example, the `cron` and `inetd` daemons are written this way.

This option is not found on any other UNIX platforms.

Daemons such as the OpenSSH daemon cannot be respawned with the `respfrk` attribute. The OpenSSH daemon also forks itself to create a child process which then continues running while the parent process ends. However, it does additional forks before actually forking the daemon process.

wait
Starts the process and waits for it to end. The process is not restarted when it ends. Any subsequent `/etc/inittab` entries are not processed until this process ends.

Command
Identifies a shell command, script, or executable program to be run. The name must be a fully qualified path name. The entire command is, by default, executed by the shell as:

```
/bin/sh -c "exec command"
```

You can change the path name of the target shell by using the `-sh` option in the `/etc/init.options` file, but if respawn is required, this action is not suggested.

You might see a message indicating that an `inittab` entry was started successfully although the command might not have run successfully. This is the case if the syntax of the `inittab` entry was correct but the command path was not a valid path name.

The /samples/inittab file

A partial sample file, `/samples/inittab`, is shown in the following example:

```
etcrc::wait:/etc/rc
inetd::respfrk:/usr/sbin/inetd  /etc/inetd.conf
msgend::once:/bin/echo Done processing /etc/inittab >/dev/console
```

Because the first entry, `/etc/rc`, does not have redirection specified, standard output and standard error are both directed to `/etc/log`.

In the sample file:

- The following entry invokes `/etc/log`:

  ```
  etcrc::wait:/etc/rc
  ```

- The following entry invokes the **inetd** daemon by means of the `/etc/initttab` file. You can use this entry instead of starting the **inetd** daemon from `/etc/log`. The sample `inittab` entry for **inetd** causes the **inetd** daemon to be started under *jobname* INETD1 with the respawn capability:

  ```
  inetd::respfrk:/usr/sbin/inetd   /etc/inetd.conf
  ```

Steps for customizing /etc/inittab

Before you begin, you need to have access to `/samples/inittab`. You should also check that the value for the `-a` option in the `/etc/init.options` file specifies enough time for the system initialization script to complete.

Perform the following steps to customize `/etc/inittab`.

1. Copy `/samples/inittab` to `/etc/inittab`. For example:

   ```
   cp /samples/inittab  /etc/inittab
   ```

   _____

2. Add entries to `/etc/inittab` for system processes that you want to be started when the system is initialized. The sample file has an entry for `inetd`. You can add any other system processes to the file. Some examples are provided:

- The following entry is defined in /etc/rc for starting the cron daemon:

  ```
  _BPX_JOBNAME='CRON' /usr/sbin/cron &
  ```

  You want to get the equivalent function using /etc/inittab. To do so, add the following entry to the /etc/inittab file:

  ```
   cron::once:/usr/sbin/cron
  ```

  The first positional, the identifier, is CRON. This will assign the job name of CRON to the started process. You do not need to set _BPX_JOBNAME. The & at the end of the /etc/rc entry is not needed in the /etc/inittab file. The & indicates the process is to be started in the background and should be forked, ensuring it gets the job name specified by _BPX_JOBNAME. Starting the process from the /etc/inittab file ensures the started process gets the job name as the specified identifier.
- To start cron the same way but with the additional function of assigning it the respawn capability, add the following entry to the /etc/inittab file:

  ```
  cron::respfrk:/usr/sbin/cron
  ```

  For commands that are only run once (during initialization), consider keeping them in /etc/rc. Consider starting daemons (long-running commands that service user requests) from /etc/inittab.

_____

3. If you have an entry for /etc/rc, remove any command entry from /etc/rc that you have added to /etc/inittab.

   _____

4. Save the file.

   _____

When you are done, you have customized the /etc/inittab file.

Customizing files for the tcsh shell

You might want to customize these files for the tcsh shell:

- /etc/csh/login
- $HOME/.login
- /etc/csh.cshrc
- /etc/complete/csrc

Customizing /etc/csh.login

The /etc/csh.login file is used for setting environment variables such as TERM and is only read by tcsh when it is a login shell.

Important: Because /etc/csh.login is the tcsh equivalent to /etc/profile for sh, you need to keep system-wide information for both sets of users in synch. Any customization that you have done for /etc/profile (such as setting environment variables) must be duplicated in tcsh syntax in /etc/csh.login. Future changes to /etc/profile also need to be made to /etc/csh.login. If you maintain a non-z/OS UNIX system, you could consider porting /etc/csh.cshrc and /etc/csh.login from that system to z/OS and merging them with the z/OS samples.

Figure 35 on page 210 shows a sample /samples/csh.login file:

```
tty -s
set tty_rc=$status
if (($?STEPLIB == 0 ) && ($tty_rc == 0)) then
    setenv STEPLIB none
    exec tcsh -l
endif
unset tty_rc

setenv TZ EST5EDT
setenv LANG C
setenv LIBPATH /lib:/usr/lib:.
setenv MAIL /usr/mail/$LOGNAME

# ============================================================
# Start of c89/cc/c++ customization section
# ============================================================
# foreach _CMP(_C89_CC_CXX)
#   setenv ${_CMP}_CLIB_PREFIX "CBC"
#   setenv ${_CMP}_PLIB_PREFIX "CEE"
#   setenv ${_CMP}_SLIB_PREFIX "SYS1"
#   setenv ${_CMP}_INCDIRS "/usr/include /usr/lpp/cbclib/include"
#   setenv ${_CMP}_LIBDIRS "/lib /usr/lib"
#
# Esoteric unit for data sets:
#   setenv ${_CMP}_WORK_UNIT "SYSDA"
# end
# unset _CMP
#
# ================================================================
# End of c89/cc/c++ customization section
#
================================================================
```

Figure 35. Partial contents of the /samples/csh.login file

Use the cp command to copy /samples/csh.login to /etc/csh.login. Edit /etc/csh.login to change or add environment variables.

Customizing $HOME/.login

To change or add environment variables such as TERM that are customized for individual users, first use the cp command to copy /samples/.login to $HOME/.login. Then edit the file to change or add environment variables. The $HOME/.login file is only read by tcsh when it is a login shell.

Important: Because $HOME/.login is the tcsh equivalent to $HOME/.profile for sh, you must keep system-wide information for both sets of users in synch. Any customization that you have done for $HOME/.login (such as setting environment variables) must be duplicated in C-shell syntax in$HOME/.profile. Future changes to $HOME/.login also must be made to $HOME/.profile. If you maintain a non-z/OS UNIX system, you could consider porting $HOME/.login and $HOME/.profile from that system to the z/OS system and merging them with the z/OS samples.

Customizing /etc/csh.cshrc

The /etc/csh.cshrc file is the system-wide profile for tcsh shell users and is read by subshells.

Figure 36 on page 211 shows suggested settings for /etc/csh.cshrc provided in the IBM-supplied /samples/csh.cshrc:

```
# ================================================================
# path shell variable
# ================================================================
#
# Specifies the list of directories that the system searches for an
# executable command.
set path = (/bin)
# ================================================================
#
#  umask variable
#
umask 022
# ================================================================
```

Figure 36. Partial contents of the /samples/csh.cshrc file

Use the **cp** command to copy the `/samples/csh.cshrc` file to `/etc/csh.cshrc`. Then edit `/etc/csh.cshrc` to change or add shell variables.

Customizing $HOME/.tcshrc

The `$HOME/.tcshrc` file contains commands that set or change the values of shell variables for individual users and is read by subshells. HOME is a variable for the path name for a user's home directory. The values set in `$HOME/.tcshrc` override those in `/etc/csh.cshrc`.

Use the **cp** command to copy `/etc`, `/var`, `/tmp`, and `/dev` to your $HOME directory. Then edit the new file to change or add shell variables.

Customizing /etc/complete.tcsh

The `/etc/complete.tcsh` file contains programmed completions that might be useful to the user. Programmed completions associate specific types of completion with individual commands.

Use the `cp` command to copy `/samples/complete.tcsh` to `/etc/complete.tcsh`. Then edit the new file.

Copying configuration files

In order to use z/OS UNIX shells and utilities, you must copy the configuration files listed in Table 31 on page 211 to the specified directory.

Table 31. Copying configuration files in order to use z/OS UNIX shells and utilities

Utility	Copied from:	To:
cron	/samples/queuedefs	/usr/lib/cron/queuedefs
file	/samples/magic	/etc/magic
inetd	/samples/inetd.conf	/etc/inetd.conf
lexx	/samples/yylex.c	/etc/yylex.c
mailx	/samples/mailx.rc	/etc/mailx.rc
make	/samples/startup.mk	/etc/startup.mk
sh	/samples/profile	/etc/profile
	/samples/.profile	$HOME/.profile

Table 31. Copying configuration files in order to use z/OS UNIX shells and utilities (continued)		
Utility	**Copied from:**	**To:**
tcsh	/samples/complete/tcsh	/etc/complete.tcsh
	/samples/csh.cshrc	/etc/csh.cshrc
	/samples/csh.login	/etc/csh.login
	/samples/.tcshrc	$HOME/.tcshrc
	/samples/.login	$HOME/.login
uucp	/samples/Devices	/usr/lib/uucp/Devices
	/samples/Dialers	/usr/lib/uucp/Dialers
	/samples/Dialcodes	/usr/lib/uucp/Dialcodes
	/samples/Permissions	/usr/lib/uucp/Permissions
	/samples/Systems	/usr/lib/uucp/Systems
yacc	/samples/yyparse.c	/etc/yyparse.c

Setting up for mesg, talk, write, and UUCP

The customization required for the `mesg`, `write`, and `talk` utilities is done at installation time. Likewise, part of UUCP customization is done at installation time. For more information, see "Security requirements for ServerPac and CBPDO installation" on page 80. In the past, these tasks were done with the FOMISCHO job from SYS1.SAMPLIB.

The FOMISCHO job remains available in SYS1.SAMPLIB for installations that cannot synchronize their security databases for the required user ID **uucp** and group IDs **uucpg** and **TTY**. To complete this customization step, these installations must run FOMISCHO against each system image.

Customizing c89, cc, and c++ (cxx) compilers

The c89 utility is customized by setting environment variables. The ones that most commonly require setting are specified in the c89 customization section.

- For the z/OS shell, the customization section is in `/etc/profile`.
- For the tcsh shell, the customization section is in `/etc/csh.login`.

c89 — Compiler invocation using host environment variables in *z/OS UNIX System Services Command Reference* lists the rest of the variables that might require setting for typical c89 usage.

z/OS UNIX System Services Command Reference, in its c89 section, assumes that the current level of the z/OS XL C/C++ compiler and Language Environment runtime library will be used. If you must use a previous level of the compiler, then you must customize other environment variables, which are only documented here.

The environment variables used by the cc command have the same names as the ones used by c89, except that the prefix is _CC instead of _C89. Likewise, for the c++ (cxx) command, the prefix is _CXX instead of _C89. Normally, you do not need to explicitly export the environment variables for all three commands; the "for loop" at the bottom of the c89 customization section can be used. This "for loop" sets the variables for all the c89/cc/c++ (cxx) commands.

By putting any customization statements for c89 into `/etc/profile` for the z/OS shell (or `/etc/csh.login` for the tcsh shell) and commenting out those lines, the environment variables are automatically exported for c89, cc, and c++ (cxx). See "Customizing /etc/profile" on page 195 and "Customizing /etc/csh.login" on page 209 for more information.

After you customize the profile, you probably will not need to change it again. However, you can change the variables at any time; the next time a user logs into the shell, they will get the new settings.

Using non-default high-level qualifiers

If any of the listed installed products did not use the installation default for the high-level qualifier, then the appropriate environment variable must be exported to make **c89** aware of this. The environment variables in this table are set to the default values for the current level of z/OS, but you will need to set them to your high-level qualifiers.

- The default value for the z/OS XL C/C++ compiler is the c89 environment variable _C89_CLIB_PREFIX=CBC.
- The default value for Language Environment is the c89 environment variable _C89_PLIB_PREFIX=CEE.
- The default value for BCP is the c89 environment variable _C89_SLIB_PREFIX=SYS1.

These high-level qualifiers are used to construct the names of data sets used by **c89**. All named data sets used by **c89** must be cataloged.

Using a system that does not have UNIT=SYSDA

If the system is not configured with an esoteric unit SYSDA, or some other esoteric unit is to be used for VIO temporary unnamed work data sets set by c89, the _C89_WORK_UNIT=SYSDA environment variable must be set. Specifying a null value for this variable ("") results in c89 using an installation-defined default for the UNIT. The environment variable used for all c89–allocated work data sets set to the default value _C89_WORK_UNIT=SYSDA.

Selecting z/OS XL C/C++ compilers

This section lists the compiler choices; the environment variable settings for each compiler are identified.

The c89/cc/c++ commands use a number of environment variables. The default values are specified as comments in the /samples/profile file that is shipped with each release. The environment variables for:

- c89 begin with the prefix _C89
- cc begin with the prefix _CC
- c++ begin with the prefix _CXX

If the C/C++ Class Library DLLS are used in building your executables (the default for the c++ command), then this will also target your executable for the same level of C/C++ Class Library

Using the same compiler for the entire system

If you are using the same compiler for the entire system, then put the compiler data set name in the linklist. By default, the linklist contains the name of the default compiler

If you are using a compiler that is not the system-wide default, then you must specify the compiler data set name in the STEPLIB environment variable and export it. Performance might be somewhat affected.

Using the command names common to the xlc and c89 utility

The c89 and xlc commands support the c89, cxx, cc, and c++ command names. Users can use the c89 or xlc versions of these commands.

The c89 utility is installed in the /bin directory. To use the c89 version of these commands, the /bin directory must precede the /usr/lpp/cbclib/xlc/bin directory in the PATH environment variable.

xlc is installed in the /usr/lpp/cbclib/xlc/bin directory. To use the xlc version of these commands, that directory must precede the /bin directory in the PATH environment variable.

If the /bindirectory precedes the /usr/lpp/cbclib/xlc/bin directory, you can still use the xlc version of these commands. To do so, use one of xlc, xlC, xlc++ and related command names (such as those with the _x and _64 suffix) and the -F option.

Example: To invoke the xlc utility using the c89 command names, issue:

```
xlc -F:c89
```

Setting up c89 to work with the current z/OS XL C/C++ compiler

These are the export statements for each compiler version, assuming that the default high-level qualifiers are being used. Where the c89 environment variables are shown, the environment variables for c++ and cc must also be set.

- For the current z/OS XL C/C++ compiler:
  - If you are using the z/OS shell, issue the following command:

    ```
    export STEPLIB="CBC.SCCNCMP"
    ```

  - If you are using the tcsh shell, issue the following command:

    ```
    setenv STEPLIB "CBC.SCCNCMP"
    ```

  Because the current z/OS XL C/C++ compiler supports compiling code that is to run on previous releases of z/OS, you do not need to use any additional c89 environment variables. All you need to do is specify the c89 option -Wc,"target(LEVEL)", where LEVEL is the level of z/OS on which the program is to be executed. For more information see TARGET in *z/OS XL C/C++ User's Guide*.
- For the IBM C/C++ V3R2 compiler:
  - If you are using the z/OS shell, issue the following commands:

    ```
    export STEPLIB="CBC.V3R2M0.SCBC3CMP"
    export _C89_CVERSION=0x13020000
    export _C89_CLIB_PREFIX=CBC.V3R1M0
    ```

  - If you are using the tcsh shell, issue the following commands:

    ```
    setenv STEPLIB "CBC.V3R2M0.SCBC3CMP"
    setenv _C89_CVERSION 0x13020000
    setenv _C89_CLIB_PREFIX CBC.V3R1M0
    ```

- For the AD/Cycle C/370 V1R2 compiler:
  - If you are using the z/OS shell, issue the following commands:

    ```
    export STEPLIB="EDC.V1R2M0.SEDCDCMP"
    export _C89_CVERSION=0x11020000
    export _C89_CLIB_PREFIX=EDC.V1R2M0)
    ```

  - If you are using the tcsh shell, issue the following commands:

    ```
    setenv STEPLIB "EDC.V1R2M0.SEDCDCMP"
    setenv _C89_CVERSION 0x11020000
    setenv _C89_CLIB_PREFIX EDC.V1R2M0)
    ```

  Because this compiler only supports the C language, it cannot be used with the c++ command.

Setting up xlc to work with the current z/OS XL C/C++ compiler

The xlc utility uses an external configuration file to control the invocation of the compiler. Before you can compile C and C++ programs, you must set up the environment variables and the configuration file

for your application. For more information about those, see xlc: Compiler invocation using a customizable configuration file in *z/OS UNIX System Services Command Reference*.

If you are using the z/OS shell, issue the following commands to set the environment variables:

```
LANG=En_US
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
PATH=/bin/c89:/usr/lpp/cbclib/xlc/bin${PATH:+${PATH}}
export LANG NLSPATH PATH
```

If you are using the tcsh shell, issue the following commands to set the environment variables:

```
setenv LANG=En_US
setenv NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
setenv PATH=/bin/c89:/usr/lpp/cbclib/xlc/bin${PATH:+${PATH}}
```

Before using the compiler, you need to install the message catalogs and set the environment variables as described in xlc: Compiler invocation using a customizable configuration file in *z/OS UNIX System Services Command Reference*.

Targeting a z/OS release earlier than the current one

The current release of Language Environment supports creating executables that will run on previous releases of the operating system. You can use the current system to build programs to run on previous releases, but the release that the program is executed on still determines what functionality is available. The C runtime library headers will detect attempts to use new function when targeting for an older releases. There is runtime detection of attempts to use new functions on all supported older releases.

Targeting an earlier release

When targeting an earlier release, you might need to pass the 'compat' option to the binder. For example:

```
-Wl,compat=min
```

A convenient way to do this as part of the setup is to use ${prefix}_OPTIONS (along with the other environment variables like ${prefix}_VERSION). Some examples are provided:

- For the z/OS shell, issue:

  ```
  export _C89_OPTIONS="-Wl,compat=min"
  ```

- For the tcsh shell, issue:

  ```
  setenv _C89_OPTIONS "-WI,compat=min"
  ```

You can set {_INCDIRS} to a null string. Some examples are provided:

- For the z/OS shell:

  ```
  export _C89_INCDIRS=" "
  ```

- For the tcsh shell:

  ```
  setenv _C89_INCDIRS " "
  ```

Or, if you have other directories that you want to be automatically searched, you can add them to {_INCDIRS}, as long as the default directories are not used with this environment variable.

Customizing the terminfo database

Full-screen application programs such as the `vi` editor and the `more` utility require a terminfo database. The terminfo database contains the characteristics of different terminal types that are used to run these full-screen applications.

The terminfo database is shipped as part of z/OS UNIX System Services Application Services. The database is populated with the terminal types defined by `ibm.ti`, `dec.ti`, `wyse.ti`, `ansi.ti`, and `dtterm.ti`. The database is in the directory `/usr/share/lib/terminfo` and the source files are in `/samples`.

If you need to define other terminal or workstations for a terminfo database, see "Steps for defining terminals or workstations for a terminfo database" on page 216.

Steps for defining terminals or workstations for a terminfo database

Before you begin, you need to know what terminals or workstation you want to define. You also need to know what directory the terminal definitions are in.

Perform the following steps to define terminals or workstations for a terminfo database.

1. Create a subdirectory in your home directory for the terminfo database terminal definition. For example:

   ```
   mkdir /u/myhome/terminfo
   ```

   where *myhome* is the name of the home directory.

   ---

2. Copy the `.ti` file for the terminal that you are building the terminfo database for into the directory that you just created. You can obtain the terminal file from another UNIX operating system, if necessary.

   For example, copy the file `pc.ti` into the directory:

   ```
   /u/myhome/terminfo/pc.ti
   ```

   where *myhome* is the name of the home directory.

   ---

3. Set the TERMINFO environment variable to the directory that the terminal definitions are in. Base your choice on the shell that you want to use.

For the . . .	Then issue . . .
z/OS shell	`export TERMINFO=/u/myhome/terminfo`
tcsh shell	`setenv TERMINFO=/u/myhome/terminfo`

   ---

4. Issue the `tic` command, specifying the terminal file. For example:

   ```
   tic /u/myhome/terminfo/pc.ti
   ```

   ---

5. Set the TERM environment variable to the name of the terminal that you want to use. Base your choice on the shell that you want to use.

For the . . .	Then issue . . .
z/OS shell	`export TERM=sun`
tcsh shell	`setenv TERM=sun`

When you are done, you have defined terminals or workstations for the terminfo database.

Re-creating the terminfo database

If you need to re-create the terminfo database, use the `tic` utility. Each type of terminal that is defined has a corresponding file with the suffix `.ti`. Some examples are provided:

1. To define an IBM terminal for the terminfo database, specify from the shell environment:

```
tic /samples/ibm.ti
```

2. To define terminal types such as VT100 and VT220, specify from the shell environment:

```
tic /samples/dec.ti
```

For information about curses, see *z/OS C Curses*.

Customizing electronic mail

The `mailx` shell command sends electronic mail between shell users on the same system.

For the z/OS shell

To enable `mailx` processing, do the following:

- Set up a system startup file, `/etc/mailx.rc`, which contains variable values and definitions common to all shell users. The IBM-supplied sample is in `/samples/mailx.rc`. Copy this file to `/etc/mailx.rc`.
- If you use a system mailbox directory other than `/usr/mail`, identify it in the $MAIL environment variable in `/etc/profile`. See "Customizing /etc/profile" on page 195.

Users can give names to mail files using variables in `$HOME/.profile` or they can use files with the default names. See "Customizing $HOME/.profile" on page 200.

For the tcsh shell

To enable `mailx` processing, do the following:

- Set up a system startup file, `/etc/mailx.rc`, which contains variable values and definitions common to all shell users. The IBM-supplied sample is in `/samples/mailx.rc`. Copy this file to `/etc/mailx.rc`.
- If you use a system mailbox directory other than `/usr/mail`, identify it in the $MAIL environment variable in `/etc/csh.login`. See "Customizing /etc/csh.login" on page 209.

Users can give names to mail files using variables in `$HOME/.login` or they can use files with the default names. See "Customizing /etc/csh.login" on page 209.

Chapter 9. Customizing for your national code page in the shell

You can set up a default language for all users of the z/OS shell. You can also customize your system so that z/OS UNIX messages are displayed in Japanese or Simplified Chinese. (They are available in English, Japanese, or Simplified Chinese.)

- For the z/OS shell, if you want to set the language for yourself, or for just one user, you can make these changes in the `$HOME/.profile`, or log on to the z/OS shell and export the LANG and NLSPATH environment variables.
- For the tcsh shell, if you want to set the language for yourself, or for just one user, you can make these changes in the `$HOME/.login`, or log on to the z/OS shell and set the LANG and NLSPATH environment variables.

For more information, see locale objects, source files, and charmaps in *z/OS UNIX System Services User's Guide*.

Lists of subtasks

Subtask	Associated procedure
Setting up your national code page	"Steps for setting up your national code page" on page 219
Customizing for Japanese and Simplified Chinese	"Steps for customizing the login file for the z/OS shell" on page 222 "Steps for customizing the login file for the tcsh shell" on page 222 "Steps for displaying messages in Japanese" on page 223 "Steps for activating MVS Message Service (MMS)" on page 223

Steps for setting up your national code page

If you will be using Japanese or Simplified Chinese, you still need to do these steps first before you continue to "Customizing for Japanese and Simplified Chinese" on page 222.

Before you begin, you need to have the login for the shell.

1. For the z/OS shell, copy `/samples/profile` to `/etc/profile`. You might have already done this, as described in "Customizing /etc/profile" on page 195.
2. For the tcsh shell, copy `/samples/csh.login` to `/etc/csh.login`. You might have already done this, as described in "Customizing /etc/csh.login" on page 209.

Perform the following steps to set up your national code page for shell users.

1. Customize the login file for your shell.

For this shell	Do this . . .
z/OS shell	Customize `/etc/profile` so that your selected national page is enabled when the shell is first invoked. Be careful that the shell, with the updated `/etc/profile`, does not keep restarting itself after you restart the shell. To make sure that `exec sh -L` is executed only once, you can copy the code in the sample `/etc/profile` and update it with your national code page.
tcsh shell	Customize `/etc/csh.login` so that your selected national page is enabled when the tcsh shell is first invoked. Be careful that the shell, with the updated `/etc/csh.login` does not keep restarting itself after you restart the shell. To make sure that `exec sh -l` is executed only once, you can copy the code that is shown in the sample `/etc/csh.login`, and update it with your national code page.

_____

2. Convert from ASCII to your national code page. Use the `chcp` command to change the data conversion for rlogin sessions.

   - For the z/OS shell, the following sample `/etc/profile` shows examples of statements to convert the terminal session data that use ASCII code page ISO8859-1 and EBCDIC code page IBM-277. This example uses the Danish locale.

```
if test -z "$LOCALE_SWITCH" && tty -s
then
  echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
  echo " - Logon shell will now be invoked to reflect        - "
  echo " - code page IBM-277                                 - "
  echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
  LOCALE_SWITCH=EXECUTED
  LANG=C
  LC_ALL=Da_DK.IBM-277
  export LANG LC_ALL LOCALE_SWITCH

 # Issue chcp if not using OMVS command
   if test "$?_BPX_TERMPATH != "OMVS" ) then
     chcp -a IS08859-1 -e IBM-277
   fi
  exec sh -L
else
  echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
  echo " -   Welcome to z/OS UNIX System Services              -"
  echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
fi
```

   - For the tcsh shell, the following sample `/etc/csh.login` shows examples of statements to convert the terminal session data that use ASCII code page ISO8859-1 and EBCDIC code page IBM-277. This example uses the Danish locale.

```
tty -s
set tty_rc=$status
if (($?LOCALE_SWITCH == 0 && tty_rc == 0)) then
  echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
  echo " - Logon shell will now be invoked to reflect        - "
  echo " - code page IBM-277                                 - "
  echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
  setenv LOCALE_SWITCH=EXECUTED
  setenv LANG=C
  setenv LC_ALL=Da_DK.IBM-277
  # Issue chcp if not using OMVS command
  if ($?_BPX_TERMPATH != "OMVS" ) then
     chcp -a ISO8859-1 -e IBM-277
     endif
     exec tcsh -l
  endif
  unset tty_rc
```

---

3. Convert these files to your selected locale by using the `iconv` command.

   - `/etc/yylex.c`
   - `/etc/mailx.rc`
   - `/etc/startup.mk`
   - `/etc/yyparse.c`

   The `lex`, `mailx`, `make`, and `yacc` utilities expect both system files and user files to be in the same code page.

   For example, to convert `/etc/mailx.rc` to be used in the Da_DK.IBM-277 locale, issue:

   ```
   iconv -f IBM-1047 -t IBM-277 /etc/mailx.rc >/etc/mailx.rc.277
   ```

---

4. Update BPXBATCH or OSHELL, if necessary.

   If you use BPXBATCH or OSHELL (which uses BPXBATCH), you must do this step in order to get the code page working immediately under BPXBATCH and OSHELL. Use the STDENV ddname to point to a file or data set that contains the environment variable definitions for the code page. The code page you specify will not affect the shell because ddname is read before the first shell is invoked, (Because the STDENV DD statement does not affect the OMVS command, you need to put the environment variables in `/etc/profile`.)

   For more information about BPXBATCH, see BPXBATCH in *z/OS UNIX System Services User's Guide*

   For more information about STDENV, see Example: Setting up code page support in a STDENV file in *z/OS UNIX System Services User's Guide*.

---

5. If you need to customize for Japanese or Simplified Chinese, go to “Customizing for Japanese and Simplified Chinese” on page 222.

---

6. If you do not need to customize for Japanese or Simplified Chinese, save the login file.

   - For the z/OS shell, it is `/etc/profile`.
   - For the tcsh shell, it is `/etc/csh.login`.

When you are done, you have set up your national code page.

To verify your code page, issue:

```
echo $HOME
```

If you entered the shell before the code page was set up, you will see $HOME. Otherwise, the shell will display the path name of your home directory. The $ should be read as your code page's dollar sign.

Customizing for Japanese and Simplified Chinese

If you are customizing for Japanese or Simplified Chinese, you need to make more changes to your login file after completing the steps in “Steps for setting up your national code page” on page 219. If you want to display your messages in Japanese or Simplified Chinese, you need to customize `/etc/init`. These changes take effect the next time OMVS is started.

You can set the system default to display translated messages. “Steps for activating MVS Message Service (MMS)” on page 223 describes the procedure.

The examples are for Japanese. Equivalent changes are required to customize for Simplified Chinese.

Steps for customizing the login file for the z/OS shell

Before you begin: You need to have `/etc/profile` set up so that you can edit it.

Perform the following steps to customize the login file for the z/OS shell so that it runs in the Japanese locale.

1. Change the line `LANG=C` to `LANG=Ja_JP`.

   _____

2. Add the following line:

   ```
   LC_ALL=Ja_JP.IBM-939
   ```

   _____

3. Enable `man` to use the `more` command as its pager:

   ```
   setenv MANPAGER /bin/more
   ```

   _____

4. Ensure that LANG and LC_ALL are specified on the line containing `export`.

   _____

5. Save `/etc/profile`.

   _____

When you are done, you have customized the login file for the z/OS shell so that it runs in the Japanese locale.

Steps for customizing the login file for the tcsh shell

Before you begin: You need to have `/etc/csh.login` set up so that you can edit it.

Perform the following steps to customize the login file for the tcsh shell so that it runs in the Japanese locale.

1. Change the line `setenv  LANG=C` to `LANG=Ja_JP`.

   _____

2. Add the following line:

   ```
   setenv LC_ALL Ja_JP.IBM-939
   ```

   _____

3. Enable man to use the more command as its pager:

```
setenv MANPAGER /bin/more
```

_____

4. Save /etc/csh.login.

_____

When you are done, you have customized the login file for the tcsh shell so that it runs in the Japanese locale.

Steps for displaying messages in Japanese

Before you begin: You need to have /etc/init.options set up so that you can edit it.

Perform the following steps to display messages in Japanese.

1. Locate the following line:

```
*e  LANG=En_US.IBM-1047
```

_____

2. Replace it with:

```
-e  LANG=Ja_JP
```

_____

3. Locate the line:

```
*e  NLSPATH=/usr/lib/nls/msg/%L/%N
```

_____

4. Replace it with:

```
-e  NLSPATH=/usr/lib/nls/msg/%L/%N
```

_____

5. Save /etc/init.options.

_____

When you are done, you have customized the /etc/init.options file to display messages in Japanese.

Steps for activating MVS Message Service (MMS)

Before you begin: Because MVS Message Service does not support translating messages to the MVS operator console, you must set up a TSO/E console that mirrors the operator's console in order to see the translated messages. TSO/E displays Japanese and Simplified Chinese messages to DBCS terminals only.

Perform the following steps to activate the MVS Message Service.

1. Compile the English and translated message skeletons.

_____

2. Create or update the following SYS1.PARMLIB members to initialize values for MVS Message Service:
   - MMSLSTxx
   - CNLcccxx
   - CONSOLxx to define the MMSLSTxx member in effect for the system

_____

3. Activate MVS Message Service.

   One way to activate MVS is to issue SET MMS=xx from the MVS operator console, where *xx* refers to the MMSLSTxx member of SYS1.PARMLIB.

   _____

When you are done, you have activated the MVS Message Service; translated messages will be displayed.

TSO/E messages

TSO/E messages are issued through MVS Message Service. For more information, see Providing translated messages in *z/OS TSO/E Customization*.

If you do not want Japanese or Simplified Chinese to be the default language, but want to see translated messages on your terminal, follow these instructions:

- For Japanese, issue PROFILE PLANGUAGE(JPN) at the TSO/E READY prompt on your DBCS terminal. This TSO/E command sets the primary language. The code JPN must match the LANGCODE statement in SYS1.PARMLIB(MMSLSTxx).
- For Simplified Chinese, issue PROFILE PLANGUAGE(CHS) at the TSO/E READY prompt on your DBCS terminal. The code CHS must match the LANGCODE statement in SYS1.PARMLIB(MMSLSTxx).

TSO/E help panels

The TSO/E help panels must be set up separately. Edit your SYS1.PARMLIB(IJKTSOxx) member in effect and ensure that the HELP statement refers to where the TSO/E help files are.

If you allocate a SYSHELP DDNAME in SYS1.PARMLIB, TSO/E searches there, rather than in the data sets pointed to by the TSO/E HELP statement.

For more information about setting up help data sets, see Specifying help data sets in *z/OS TSO/E Customization*.

Concatenating target libraries to ISPF

To use the Japanese translation of the panels, messages, and tables, you must concatenate the following target libraries to the appropriate ISPF data definition names (ddnames):

- SYS1.SBPXPJPN to ISPPLIB
- SYS1.SBPXMJPN to ISPMLIB
- SYS1.SBPXTJPN to ISPTLIB
- SYS1.KHELP to SYSHELP

To use the Simplified Chinese translation, concatenate the following target libraries to the appropriate ISPF ddnames:

- SYS1.SBPXPCHS to ISPPLIB
- SYS1.SBPXMCHS to ISPMLIB
- SYS1.SBPXTCHS to ISPTLIB
- SYS1.PHELP to SYSHELP

PROFILE PLANGUAGE and the OMVS command

The PROFILE PLANGUAGE setting in effect when the OMVS TSO/E command is first issued determines the language for all OMVS command messages not from TSO/E, until you exit OMVS and return to TSO/E.

If PROFILE PLANGUAGE(JPN) is specified, and later you go to TSO/E and enter PROFILE PLANGUAGE(ENU), most TSO/E messages appear in English—including TSO/E messages about the OMVS command.

However, any OMVS command message not from TSO/E (such as the help panels invoked from <PF1> or the FSUM23-prefix messages) is displayed in Japanese. In particular, the TSO/E prompt message "`OMVS - enter a TSO/E command`" is still displayed in Japanese but all other messages are displayed in English while you are in TSO/E.

Chapter 10. Configuring the UNIX-to-UNIX copy program (UUCP)

UNIX-to-UNIX copy program (UUCP) is a group of programs, directories, and files that can be used to communicate with any UNIX system that is running a version of the UNIX-to-UNIX copy program (UUCP). A UUCP network traditionally consists of a group of computers joined in a network using serial connections or TCP/IP. The z/OS UNIX implementation of UUCP uses TCP/IP; it does not provide modem support. It is also XPG4-compliant.

The UUCP functions are used to automatically transfer files and requests for command execution from one UUCP system to another, typically in batch mode at scheduled intervals. You can use UUCP for file transfer, remote command execution, and custom applications.

If you use a UUCP utility to transfer a file or execute a remote command, a job request is created and queued. Depending on how UUCP has been configured at your system, the job might be processed immediately or remain queued and only be processed at scheduled times. At some point, either your system will contact the remote system, or be contacted by the remote system at which time the queued jobs will be processed. For security purposes, configuration files on each system control which transfers can take place and which commands can be executed. (See "Create or edit UUCP configuration files" on page 234.)

You must decide if you want to have your system participate in a UUCP network. If you already have made that decision, go to "Configuring your local system" on page 231.

Restriction: UUCP is restricted to an 8-character password. It does not support password phrases.

Transferring files

UUCP can send and receive files between systems. The `uucp` command queues requests for file transmission or retrieval, and invokes `uucico` to establish the connection with the remote system and complete the transfer. Based on configuration specifications, file transfers with the remote system might not be allowed. The **`cron`** daemon can be used to invoke `uucico` to send the queued files in the background when appropriate. After `uucico` has made a connection with a remote system, and local `uucp` requests have been processed, file transfer requests created on the remote system are processed.

Executing commands from a remote location

The `uux` command allows you to run a program at another system, with the appropriate permissions. An execute file is sent to the remote system where it is treated as a command (like a batch file).

Tailoring UUCP for custom applications

You can also tailor UUCP for custom applications, which can send or collect data from remote systems and execute commands remotely. A common application built on UUCP is public discussion groups, called *netnews*, or simply *news*. The net is a public forum (consisting of member systems) for the exchange of ideas in the form of news articles. Users belonging to the member systems can post, read, and reply to news.

UUCP commands and daemons

UUCP provides the `uucp` command, which schedules files to be exchanged with other UUCP systems, and the `uux` command, which schedules commands to be executed by other UUCP systems. However, the `uucp` and `uux` commands do not cause any files to be exchanged or commands to be executed. For this, UUCP provides two daemons called `uucico` and `uuxqt` which establish communication sessions, transfer data, and execute commands according to the requests scheduled by `uucp` and `uux`.

The commands associated with UUCP are:

uucc
Compile UUCP configuration files

uucp
Copy files between remote systems

uulog
Display log information about UUCP events

uuname
Display a list of UUCP systems

uupick
Manage files sent to you via `uuto`

uustat
Display the status of pending UUCP transfers

uuto
Copy files to users on remote systems

uux
Request command execution on remote systems

The daemons associated with UUCP are:

cron
Starts the `uucico` daemon according to the schedule specified

inetd
This TCP/IP daemon starts the `uucpd` daemon

uucico
Processes `uucp` and `uux` file transfer requests

uucpd
Invokes the `uucico` daemon for TCP/IP connections from remote `uucp` systems

uuxqt
Run commands from other systems

UUCP directories and files

The directories associated with UUCP are:

- `/usr/spool/uucppublic`, the public UUCP directory that is the default directory for storing files that have been transferred to the local system by `uucp`.
- `/usr/spool/uucppublic/receive`, a subdirectory in the public directory for files sent from remote systems using `uuto`.
- `/usr/spool/uucp`, the spool directory that holds all work requests and all log files for UUCP.
- `/usr/spool/uucp/.Sequence`, a subdirectory for sequence files used by `uucp` and `uucico`.
- `/usr/spool/uucp/.Status`, a subdirectory containing status files for each remote system.
- `/usr/spool/uucp/.Xqtdir`, the working directory for `uuxqt`.
- `/samples`, the directory that sample configuration files are shipped in.
- `/usr/lib/uucp`, the directory that customized configuration files reside in.

As of z/OS V1R13, `/usr/spool` is configured as a symbolic link to `/var/spool`. Also, the configuration files now reside in `/var/uucp` with symbolic links directed from `/usr/lib/uucp` for each file. The setup commands in this chapter can still be performed using the original root directory as documented, whereby the symbolic link will redirect them as necessary. See "Customizing the cron, uucp, and mail utilities for a read-only root file system" on page 123 for more information.

For a discussion of configuration files, see "Create or edit UUCP configuration files" on page 234.

For a discussion of system files, see “Log files, lock files, status files, and working files” on page 247.

The UUCP communications network

A UUCP network consists of a number of systems that exchange information. Each system has a working copy of UUCP and a unique name that identifies it in the network. There is no central control system for a UUCP network; each system controls its own connections. In a UUCP network, computers connect computers in the same building or to networks that include computers around the world.

In a UUCP network, every system (also known as a site) communicates with at least one other system, but does not have to call all the sites. See Figure 37 on page 229 for a diagram of a simple network. This network has four sites that are named North, South, East, and West.

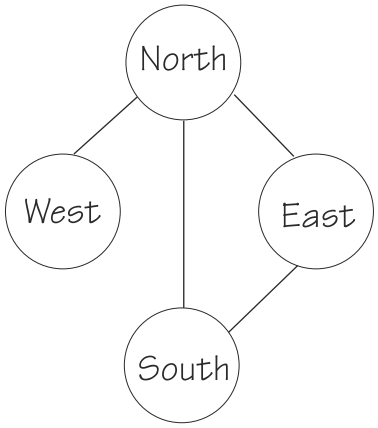


Figure 37. A simple UUCP network

The lines indicate a direct connection between two systems:

- North connects to East, South, West.
- East connects to North, South.
- South connects to North, East.
- West connects to North.

Each system exchanges files with the systems it calls directly. Users on North can send files directly to any of the other three systems, but users on West can only send files directly to North. These are called direct connections to distinguish them from connections made through intermediaries. Someone on West can send a file to someone on East indirectly through North, if North has agreed to pass along file requests from West to East. This makes North an intermediary node.

Alternatively, North could set up its configuration so that West could not transfer files through North, but only to North. This is called a *terminal* or *leaf-node* connection. For information about defining connections between two nodes, see the `COMMANDS` option in “The Permissions file” on page 239.

Securing your system

UUCP gives users on other systems access to your computer. By default, remote users can only write data to your public directory; they cannot read any data nor can they execute any commands.

However, remote users potentially could copy files to your file system or from your file system. They could also run commands on your system. How do you ensure that they do not remove files you want, read your private files, or run commands that damage your system? In short, how do you keep your UUCP system secure?

There are three things to consider in security:

- Authorization—Who is authorized to access your system?
- Access—What files can users on other systems read and write?
- Execution—What commands can users on other systems run on your system?

Authorization is the highest level of security. Only those with the current NUUCP password can access your system and even then, only authorized systems can use it. There is one catch, however, and that is when more than one system is involved in the file transfer (a multi-hop transfer). If South allows North access, there is nothing South can do to prevent North from giving West the ability to use North as an intermediary node between South and West. South cannot differentiate between requests originating from North and requests being forwarded through North.

To deal with the security issues of access and execution, UUCP uses the concept of *permissions*. For each directly connected system, you assign access permissions to look at a specific portion of your file system and execute permission to run certain commands.

Permissions can be broad or restrictive. If you are using UUCP to connect a group of machines in your office, you might want everyone to have access to all the files and be able to run all of the commands on each machine. On the other hand, you might not want private files to be made public.

For example, imagine a central office with many branch offices. The central office uses remote commands to run reports in each branch office, and send the results back to the central office. The central office needs permission to run the command that produces reports, and it needs permissions to read and write the associated files. People on other systems do not need those files or permissions. In fact, it could be dangerous to the company to allow those permissions to anyone else.

If one of the branch offices has a connection to a different UUCP network, private information could go out worldwide. The branch office denies that outside connection permission to run any commands which produce reports or to read those files. It limits the outside system to reading and writing in a small part of the file system, perhaps one directory. This directory is the only part of the file system that all other UUCP systems can read or write — it's public. Not surprisingly, this directory is called the *public UUCP directory*.

The public UUCP directory

The public UUCP directory is the default destination for files that have been transferred to the local system from other systems. Additionally, if a remote user has read access to the local system, by default the directory he can read from is the public directory. The files remain in the public directory until users claim them.

Typically, users on the local system have read access to the public directory (and sometimes write and execute access as well). So users can access files in the public directory using normal file access methods (for example, `cp`, `cat`, or `vi`)—or for files sent by `uuto`, they can use `uupick` to handle them. The `uuto` command, a simplified method of using `uucp`, uses the `receive` subdirectory of the public directory as its target. Within that subdirectory, each user on the local system has a subdirectory.

The public UUCP directory is called `/usr/spool/uucppublic`, and it is created when the BPXISMKD job is run as part of the z/OS installation. (Other UUCP systems or operating systems might use different names for the public directory.) Within the public directory, UUCP creates a subdirectory for each remote system that sends files to the local system.

As of z/OS V1R13, `/usr/spool` is configured as a symbolic link to `/var/spool`. The public UUCP directory must reside in the `/var` directory and the subdirectories created by UUCP are created in `/var/uucppublic`. See “Customizing the cron, uucp, and mail utilities for a read-only root file system” on page 123 for more information.

To make file transfers easier, you can use a special character in path names for the public UUCP directory: when tilde (`˜`) is written as the first directory in a destination path name, the `˜/` stands for the public UUCP directory. You can specify the public UUCP directory with the path name `˜/`. The public UUCP directory is defined as the home directory of the user `uucp`, so you can also specify it as `˜uucp`.

Execute permissions

By default, UUCP does not give permission for remote systems to run any commands; you must specify the commands that remote systems can run at the local system.

If you are willing to pass along files copied with uucp, or if you want to allow other direct systems to use wild cards when requesting files from your system, give the remote system permission to run the uucp command. If you do not assign execute permissions for the uucp program to another system, it can transfer files only to your system (meaning your system is a terminal node, or leaf-node) but not through your system (meaning your system is an intermediary node).

If a remote system has permission to run uucp, it can be an intermediary for another system to which it is connected. Your local system cannot distinguish if an incoming request from an authorized system originated at that system or at a system to which it is connected. Therefore, you must assume that the execute permission you give to a remote system can be inherited, or used, by another system to which it is connected.

To use wild cards to request files from your system, the remote system must have permission to run uucp and also have read permission on the directory holding the files.

- For an example of a multiple system uucp transfer, see uux - Request command execution on remote UUCP systems in *z/OS UNIX System Services Command Reference*.
- For an example of how to give a remote system permission to run uucp, see "The Permissions file" on page 239.

Configuring your local system

To configure your local system for UUCP access, you must: take the following actions

1. Determine your local system name.
2. Create or edit configuration files.
3. Define the new user ID, NUUCP, to RACF. The other required user ID, *uucp*, and group ID, *uucpg*, were already defined at installation time.

Determine your local system name

To determine what your system is called in the shell, issue:

```
uname -n
```

You will see the name by which your system is known in a communications network. It is the name specified by the IPL parameter SYSNAME. In z/OS UNIX, UUCP recognizes the first eight characters of this name. Other UNIX systems might recognize more or fewer characters.

Add an entry to the permissions file

UUCP uses five different configuration files to describe various aspects of your UUCP setup. (To learn more about the configuration files, refer to "Create or edit UUCP configuration files" on page 234 before proceeding with this section.)

The Permissions file is used to control the access that remote systems have to data and programs on the local system. You might want to change some of the default settings of the Permissions file.

If you need to change some of the default permissions for your local system (such as PUBDIR, READ, WRITE, NOREAD, or NOWRITE) then you will need an additional entry in the Permissions file for your local system. If you do not need to change the default permissions then you do not need an entry in the Permissions file for your local system.

For example, if you wanted to change your uucp public directory, your Permissions file might look like this:

```
    MACHINE=local                \
           READ=/readall         \
           PUBDIR=/free
    MACHINE=site1:site2:SITE3    \
           READ=/readall         \
           COMMANDS=uucp:cat:cp:ls
    LOGNAME=NUUCP                \
           READ=/readall         \
```

```
          PUBDIR=/free           \
          SENDFILES=yes          \
          VALIDATE=site1:site2:SITE3
```

Define the group ID and the user ID to RACF

As a customization step for UUCP, a UUCP-specific group ID (*uucpg*), and at least two user IDs are defined. The user IDs are:

- *uucp*, the user ID that owns all the UUCP files and directories. Use it when editing configuration files or performing other administrative tasks. The user ID `uucp` and group ID `uucpg` are now requirements for Portable Software Instance and CBPDO installations. See "Security requirements for ServerPac and CBPDO installation" on page 80.
- A LOGNAME user ID that remote systems use when dialing in to your system. Traditionally, this user ID begins with NUUCP. For purposes of example here, we use NUUCP as the user ID. You might want to establish more than one LOGNAME user ID to handle different levels of access for remote systems.

You need to define these IDs to RACF. (If you are using an equivalent security product, refer to that product's documentation for more information about defining IDs to the security product.) All the RACF commands are issued by a TSO/E user ID with RACF SPECIAL authority. To make it easier to transport data sets from test systems to production systems, duplicate these entries in all of your security data bases, including the same UID and GID values in the OMVS segment.

If you use only uppercase IDs on your system, follow these steps to define the group ID and user IDs:

1. To define the LOGNAME user ID (in this example, it is specified as NUUCP), issue the following command:

   ```
   ADDUSER NUUCP DFLTGRP(UUCPG) PASSWORD(xxxxxxx)
   OMVS(UID(397) HOME('/usr/spool/uucppublic')
   PROGRAM('/usr/lib/uucp/uucico'))
   ```

   where:

   - 397 is an example of a unique UID. Do not use UID(0).
   - `HOME('/usr/spool/uucppublic')` is a required parameter that specifies the initial path name for the directory.
   - `PROGRAM('/usr/lib/uucp/uucico')` is a required parameter that specifies the initial path name for the shell program.
2. Consider defining other user IDs similar to NUUCP to provide different access to your systems resources to the different remote systems issuing requests to your system. Each would have a unique UID, but would have the same attributes as NUUCP. In particular, each must have home directory of `/usr/spool/uucppublic` and initial program of `/usr/lib/uucp/uucico`. The UUCP permissions file is used to specify what these user IDs can access, as explained in "The Permissions file" on page 239.

Also follow these steps if you already use mixed-case group and user IDs on your system and the users do not conflict with existing names. You might want to add the lowercase names to your alias table, mapping them to uppercase names. This is not necessary, because when the lowercase names are not found in the alias table, they are folded to uppercase. For more information about the alias table, see "USERIDALIASTABLE" on page 33.

If a name such as NUUCP is not allowed on your system (or if it conflicts with an existing name), these are the RACF commands to define the user ID.

To define a LOGNAME user ID of *xxnuucp*):

```
ADDUSER xxnuucp DFLTGRP(UUCPG) PASSWORD(xxxxxxx)
OMVS(UID(397) HOME('/usr/spool/uucppublic')
PROGRAM('/usr/lib/uucp/uucico'))
```

where: *xxnuucp* is replaced by a 1- to 7-character user ID of your choice. This is the user ID that remote systems use when communicating with your system.397 is an example of a unique UID. Do not use

UID(0). HOME('/usr/spool/uucppublic') is a required parameter that specifies the initial path name for the directory. PROGRAM('/usr/lib/uucp/uucico') is a required parameter that specifies the path name for the shell program.

You might want to define other user IDs similar to NUUCP to provide different access to your system resources to the different remote systems issuing UUCP requests to your system. Each would have a unique UID, but would have the same attributes as NUUCP. Each must have home directory of /usr/spool/uucppublic and initial program of /usr/lib/uucp/uucico. The UUCP Permissions file is used to specify the accessibility of each of these user IDs.

Define an alias for the xxnuucp user ID in your user ID and group name alias table.

```
xxnuucp nuucp
```

Tip: Using the alias table causes poorer performance and increases systems management costs and complexity. For more information about the alias table, see "USERIDALIASTABLE" on page 33.

Configuring communication with remote systems

To configure UUCP so that it can communicate with remote systems, you must establish the appropriate communication protocols in the Systems file and create working directories for each supported remote system. To do this:

1. Obtain information about remote systems.
2. Create or edit the required configuration files.
3. Compile the configuration files with the uucc utility.
4. Create working directories for local and remote systems.
5. Schedule UUCP transfers with cron.

Obtain information about remote systems

Before attempting remote system configuration, contact the system administrator for that system. Together, you must decide if your system can call the other system, the other system can call your system, or either system can call each other.

If your system is going to call the remote system, you need the following information:

- The UUCP name of the remote system.
- A UUCP login account on that system. You need a login name and a password, so your system can log into the remote system.
- The login procedures. Ask whether any special send/expect sequences are used. Send/expect sequences are explained in the description of chat scripts in "The systems file" on page 234. Alternatively, you can attempt a remote login and note the sequence of prompt strings and commands required to login.
- You might need scheduling information, because there might be times when UUCP connections are not allowed. You can control the time when calls are made—for example, when lower rates are available—either with the scheduling information in the Systems file or when setting up cron.

If the remote system will call your system, you must provide the following information:

- Your system name.
- A login name for the remote system.
- A UUCP password for the remote system.
- The send/expect sequence. For z/OS systems, this would typically be in the following format:

```
in:--in: uucp_login_user password: password
```

where:

- `uucp_login_user` is the login user ID that the remote system is authorized to use—such as NUUCP.
- `password` is the password for the login user ID.

Create or edit UUCP configuration files

UUCP uses five different configuration files to describe various aspects of your UUCP setup. Sample configuration files are shipped in the `/samples` directory. You can customize them. To do this, log in as the UUCP user or use the **su** command to switch to the user uucp so that you can create or change configuration files. Copy each configuration file into the `/usr/lib/uucp` directory and customize it with entries for your local system. Table 32 on page 234 lists these files and summarizes the information each one contains. Because the UUCP configuration files are delivered as symbolic links that are directed to files in the `/var/uucp` directory, you can copy the files directly to `/var/uucp` if you prefer. See "Customizing the cron, uucp, and mail utilities for a read-only root file system" on page 123 for more information.

Table 32. UUCP configuration files. List of UUCP configuration files and their contents

File	Contents
Systems	Lists each supported system and describes when and how to establish a connection for each system. See "The systems file" on page 234.
Devices	Describes communications hardware on your system. See "The Devices file" on page 238.
Dialers	Contains dialing instructions for your system's modems. See "The Dialers file" on page 239.
Dialcodes	Defines abbreviations that can be used as part of phone numbers. See "The Dialcodes file" on page 239.
Permissions	Defines for each remote system the sections of your file system that can be read from or written to and the commands which can be executed on your system by that system. Also defines how your system exchanges queued work with remote systems. See "The Permissions file" on page 239.

If you need to make changes to your configuration, edit the configuration files and then run the `uucc` command to compile them.

Editing a configuration file

If you need to extend an entry in the configuration file over two or more lines, end all lines, except the last one, with a backslash (\).

Look at this Permissions file, for example:

```
LOGNAME=uwest  MACHINE=west  READ=/  WRITE=/       \
  COMMANDS=uucp:cat  NOREAD=/usr/private   \
  NOWRITE=/usr/private  SENDFILES=yes  REQUEST=yes \
  VALIDATE=west

LOGNAME=nuucp  MACHINE=OTHER  REQUEST=yes  \
  SENDFILES=call
```

The systems file

The Systems file contains at least one entry for each remote system that your system is going to call. It provides information for the `uucico` utility to use when it is invoked. Each entry in the file has the format:

```
system  sched  device_type  speed  phone  chat_script
```

For example,

```
sys2 Any TCP - sys2.kgn.ibm.com  in:--in: nuucp ssword: uupasswd
```

The following list describes what each field represents:

system
The name of a remote system. This name must be unique (compared to other remote system names in the Systems file) in its first seven characters.

sched
The times when your local system is permitted to call *system*. There are four subfields in the `sched` field: day, time, grade, and retry.

An example `sched` field looks like this:

```
Mo1200/C;5
```

where `Mo` is the `day` subfield, `1200` is the `time` subfield, `/C` is the `grade` subfield, and `5` is the `retry` subfield.

The description of each subfield follows:

day
Indicates which days of the week your system can call the remote system named by *system*. The abbreviations `Mo`, `Tu`, `We`, `Th`, `Fr`, `Sa`, and `Su` represent individual days. You can also use the following keywords:

Any
Your system can call the remote system on any day.

Never
Your system should never call the remote system. It should only wait to be called.

Wk
Your system can call the remote system on any weekday (that is, Monday-Friday).

time
The range of times during which your system can call the remote system named by *system*. This subfield immediately follows the `day` subfield with no intervening spaces. The times given apply only to days specified by `day`. If you do not specify a time subfield, your system can call the remote system any time during the given days. The format of this subfield is:

```
time1-time2
```

where both `time1` and `time2` are 24-hour clock times. For example,

```
WeTh0730-1415
```

means that your system can call the remote system between 7:30 a.m. and 2:15 p.m. on Wednesdays and Thursdays. This time range can extend over 0000 (midnight), but be careful. It doesn't quite work the way you might expect it to. For example,

```
Mo2300-0700
```

does not indicate 11:00 p.m. on Monday through 7:00 a.m. on Tuesday, but rather midnight through 7:00 a.m. on Monday morning and 11:00 p.m. through 11:59 p.m. on Monday evening.

You can specify multiple `day`/`time` combinations in an entry by separating them with a comma. For example a Systems file entry containing

```
Th0800-1600,Fr1215-1900,SaSu
```

indicates your system can call the remote system during the following times:

- 8:00 a.m. through 4:00 p.m. on Thursday
- 12:15 p.m. through 7:00 p.m. on Friday
- Anytime on Saturday or Sunday

grade
An optional subfield that lets you specify the minimum grade of work file that `uucico` will send during a given time period (as indicated by the `day/time` subfields). A grade is a single digit, or a single uppercase or lowercase letter. In order of priority, from highest to lowest, the grades are arranged

```
0  1  2  …  9  A  B … Z  a  b  …  z
```

That is, `0` has the highest priority and `z` has the lowest).

As work files are created for UUCP file transfers, they are automatically assigned grades that determine the order in which they are sent. By default, `uux` requests have a grade of `A` and `uucp` requests have a grade of `n`.

This optional subfield is separated from a day/time pair by a slash. For example,

```
MoTu0800-1200/C
```

indicates that only work files with a grade of `C` or higher will be sent during the hours of 8:00 a.m. to noon on Mondays.

The `grade` subfield only controls outgoing files during the given time period. It does not affect incoming files.

retry
An optional subfield that indicates how many minutes after an unsuccessful call to a remote system, `uucico` should wait before trying to call that system again. This subfield, if specified, appears at the end of the `sched` field (separated by a semicolon). For example, a `sched` field of

```
Any;60
```

indicates that your system can try to call the remote system at any time and if it is not successful in connecting, it will not try again for 60 minutes.

If you do not include the `retry` subfield, `uucico` waits five minutes after the first unsuccessful connection attempt. This waiting period doubles after each subsequent failure.

device_type
Only TCP/IP connections are supported, so specify TCP.

speed
Only TCP/IP connections are supported, so specify – (hyphen).

phone
Only TCP/IP connections are supported, so this field must contain the IP address of the remote system, or a host name by which the IP address is known. You should be able to ping this address. For example, from TSO/E:

```
ping omvsoe2a
PING V3R1: PINGING HOST OMVSOE2A  (198.151.241.130).  USE
PING: PING #1 RESPONSE TOOK 0.004 SECONDS. SUCCESSES SO FAR
```

chat_script
A text string that defines the initial login conversation that takes place between your system and the remote system. It has the format:

```
expect_string send_string expect_string send_string …
```

where `expect_string` is the text string that you expect to receive from the remote system and `send_string` is the text_string that you want to send in response. These two strings are separated by blanks. For example, when you login to a remote system, it responds with

```
login:
```

Type

```
nuucp
```

and press ENTER. The remote system then replies

```
password:
```

Enter your password

```
Shazam!
```

and press ENTER.

This conversation can be expressed as the following chat script:

```
login: nuucp password: Shazam!
```

This chat script tells `uucico` to expect the string *login*. After it is received, reply by sending the string *nuucp* (automatically sending a newline afterwards). `uucico` then waits for the string *password* and replies with `Shazam!`.

The *expect_string* can be any part of the string expected from the remote string. Thus, the sample chat script could be written:

```
ogin: nuucp ssword Shazam!
```

and yield the same result.

Tip: Omit the first letter of the login and password because some systems might use capital letters for one or both of the words and some might not. To avoid having to find out which way a system is and possible changes on the remote system, the first letters are omitted from the expect string.

The *expect_string* can be replaced with a string of the format:

```
expect_string-subsend_string-subexpect_string
```

where *subsend_string* and *subexpect_string* are text strings similar to *send_string* and *expect_string*. Hyphens separate the *expect_string*, the *subsend_string*, and the *subexpect_string*. With this format, your system waits for *expect_string* from the remote system and if it is received within a reasonable length of time, `uucico` responds with the *send_string*. If it is not received, `uucico` sends the *subsend_string*, waits for the *subexpect_string*, and then finally sends the *send_string*.

For example, if you were using a chat script and there was noise on the line that garbled the `login:` string, the chat script would fail. However, the following chat script might work:

```
ogin:--login: nuucp ssword: Shazam!
```

This script waits for `login:` from the remote system. If it is not received, `uucico` replies by sending a null string (there is nothing between the two hyphens) followed by a newline. `uucico` then again waits for `login:`. When it is received, `nuucp` is sent. The remainder of the script is identical to the earlier example.

If you want an *expect_string* to wait a specific length of time for a match, you can suffix the *expect_string* with a tilde (˜) followed by a number. The number is the number of seconds to wait for the *expect_string*. For example, the chat script

```
ogin:&tilde;10--login: nuucp ssword: Shazam!
```

waits 10 seconds for the string `login:` before continuing. You can use this suffix with *subexpect_string* as well.

Table 33 on page 238 shows the escape sequences which you can use in a chat script.

Table 33. Escape characters that can be used in chat scripts	
Escape	**Description**
""	Expect a null string
EOT	Send the end-of-transmission character
BREAK	Cause a BREAK
\b	Send a BACKSPACE
\c	Suppress newline or carriage return
\d	Delay for one second
\K	Send a BREAK
\n	Send a newline
\N	Send a NULL
\p	Pause for a fraction of a second
\r	Carriage return
\s	Send a space
\t	Send a tab
****	Send a backslash
\ ˜	Expect a tilde
ddd	Send the EBCDIC character with octal code *ddd*. For example, use `\100` to represent a space character.

The Devices file

The Devices file contains information for direct links and network connections. Each entry has the format:

```
type  dataport  – speed  dialer-arg …
```

Only TCP/IP connections are supported. Specify this entry in the Devices file:

```
TCP - - -
```

The following list describes what each field represents:

type
Describes the type of link. The following keyword is a valid entry for this field:

TCP
A link through TCP/IP. When TCP is specified, each of the other fields should contain a hyphen. You must also specify TCP in the `dialer` field of the Dialers file.

dataport
Is the device name of the port used to make the connection. For TCP/IP, specify a – (hyphen).

–
An obsolete field, *dialer-port*. Specify a hyphen.

speed
This speed must match the value of the speed field in the corresponding Systems file entry. For TCP/IP, specify a – (hyphen).

dialer-arg
Contains a list of one or more pairs with the format

```
dialer  arg
```

where *dialer* is the dialer name and *arg* is an argument to pass to that dialer. For TCP/IP, leave this blank.

The Dialers file

The Dialers file provides dialing instructions for the dialers referred to in the Devices file. Each entry has the following format:

```
dialer  subs  expect-send  [expect-send …]
```

Only TCP/IP is supported, so the entry that must appear in this file is:

```
TCP
```

The following list explains what each of these fields represents:

dialer
The type of dialer. For TCP/IP, specify TCP.

subs
Contains a string of characters. For TCP/IP, leave this blank.

expect_send
Is similar to the *chat_script* field in the Systems file. The major difference is the set of escape characters which can be used.

The Dialcodes file

The Dialcodes file contains abbreviations that can be included in the phone numbers specified in the Systems file. For TCP/IP, in this file you specify:

```
-  -
```

The Permissions file

The Permissions file is used to control the access that remote systems have to data and programs on the local system. Specifically, it is used to specify:

- Which systems can establish a `uucico` connection.
- The areas in the file system that a remote system can read or write from.
- The commands that the remote system can run on the local system.
- If the local system will process its waiting work when contacted by another system.
- An alias for the local system.
- A different public directory.

The format of each entry in the Permissions file is:

```
LOGNAME=userid [MACHINE=system] option=value  [option=value] …
```

or

```
MACHINE=system [LOGNAME=userid] option=value  [option=value] …
```

where *option* is one of the options and *value* is one or more values that you want to set for that option. Options and values are case-sensitive. When specifying multiple values for an option, separate the values with a colon (:). Here is a sample entry:

```
MACHINE=ME READ=/ WRITE=/ COMMANDS=ALL
MACHINE=site1:site2:SITE3       \
        READ=/                   \
```

```
        WRITE=/                  \
        COMMANDS=uucp:cat:ls
LOGNAME=NUUCP  \
        READ=/                   \
        WRITE=/                  \
        SENDFILES=yes            \
        DEBUG=9                  \
        VALIDATE=site1:site2:SITE3
```

The Permissions file can also contain blank lines (which are ignored) and comment lines. To indicate that a line is a comment line, use a number sign (#) as the first character in the line.

Each entry must contain the LOGNAME option or the MACHINE option, or both. Both options are used to identify an entry that applies to a remote system when it is processing its file transfer requests. The difference between them is based on which system initiates the connection:

- LOGNAME=userid entries apply to a remote system when it initiates the connection by logging onto your system as *userid*.
- MACHINE=system entries apply to a remote system when your system initiates the call to *system*.

If your system initiates the connection, your system first processes any queued file transfer requests that it has. When this is complete, the remote system can indicate that it has file transfer requests queued on its system that it would like to process. If the correct permissions are set, control switches to the remote system which then processes its file transfer requests. At this point, the MACHINE entry options are used for the remote system.

If your system does not need to differentiate Permissions options based on which system initiates the call, then LOGNAME and MACHINE can appear in the same entry.

These are the LOGNAME and MACHINE options:

LOGNAME
Indicates the user IDs that remote systems can use when logging on to your system. For z/OS systems, these names must be specified in uppercase unless USERIDALIASTABLE is used to define lowercase or mixed-case aliases. See "USERIDALIASTABLE" on page 33 for more information about defining user aliases.

MACHINE
Specified as MACHINE=system, this indicates the remote systems that your system can call using the other options specified in this entry. The system name specified here must also be specified as a system in the systems file. If you set this option to OTHER, the options specified apply to any remote system not specified by a MACHINE option in another entry. For remote systems, these names are typically uppercase. Contact the remote system's UUCP administrator to make sure that the names are uppercase.

Permissions for uux commands (which are executed by uuxqt) are based on MACHINE entries regardless of which system initiates the call.

These are the valid options that are used with either LOGNAME or MACHINE entries, or with both. Options are marked with an (L) or an (M) to indicate that they are intended for LOGNAME or MACHINE entries or for both (L,M). An option used in an entry for which it is not intended will be ignored.

READ
(L,M) Indicates which directories uucico can read. By default, this is the home directory of user uucp (/usr/spool/uucppublic). Remember that uucico runs with the effective UID of UUCP, so you must permit the *uucp* user or *uucpg* group to read from these directories.

WRITE
(L,M) Indicates which directories uucico can write to. By default, this is /usr/spool/uucppublic, the home directory of user uucp. Remember that uucico runs with the effective UID of UUCP, so you must permit the *uucp* user or *uucpg* group to write to these directories.

NOREAD
(L,M) Indicates that files in the specified directories cannot be read. If a directory is specified by both READ and NOREAD, files in that directory cannot be read. The public directory can always be read (even if specified on NOREAD).

NOWRITE
(L,M) Indicates that files in the specified directories cannot be written to. If a directory is specified by both WRITE and NOWRITE, files in that directory cannot be written to. The public directory can always be written to (even if specified on NOWRITE).

PUBDIR
(L,M) Indicates the public directory. By default, this is the home directory of user `uucp` (`/usr/spool/uucppublic`).

If you are going to change PUBDIR on your system, you need to have an additional MACHINE entry for your local site. Consider this example:

```
uucp remote_site!/file1 local_site!&tilde;/file1
```

When `uucp` processes this command it looks for a `MACHINE=local_site` entry to find the value for `PUBDIR`.

DEBUG
(L,M) Indicates the verbosity of the debugging information. Set this to a number between 0 and 9. Level 0 provides terse debug messages while level 9 provides verbose output. This output is stored in `/usr/spool/uucp/LOGFILE` to aid you in debugging communications problems when remote systems call you.

REQUEST
(L,M) Indicates whether requests made by remote systems to transfer data from your system are allowed. This option can be used to protect data on your system from being read by remote systems.

- If set to `yes`, remote systems can read data from those directories it is authorized to read from.
- If set to no, a remote system can write data to your system, but cannot read data irrespective of the value of the READ option. This is the default.

This option only applies to requests originating from the remote system. This option has no effect on file transfer requests that originate on your system.

SENDFILES
(L) Indicates if your system will process its own queued file transfer requests after the remote system has initiated the connection and completed its file transfer requests. The SENDFILES option allows the local system to control when its queued file transfer requests are processed.

- If this option is set to `yes`, your system will process its queued requests after the remote system has completed processing its own.
- If this option is set to `call`, your system will only process its own file transfer requests when it initiates the connection with the remote system. This is the default.

VALIDATE
(L) Names the remote systems that can login to your system using the user IDs given by `LOGNAME`. If another system attempts to login using this user ID, `uucico` refuses the connection.

COMMANDS
(M) Indicates the commands that the remote system can execute on your system.

By default, the `uucp` command is not permitted, which means that by default your local system is a terminal, or leaf-node, connection. To allow a remote system to transfer files through your local system, specify `uucp` for the `COMMANDS` option.

- To specify more than one command, separate the command names with a colon (:). For example, `COMMANDS=uucp:ls`.
- To prohibit all commands, do not use the `COMMANDS` option.
- To allow access to all commands, set this option to `ALL`.

MYNAME
(M) Tells the remote system that the name of your local system is the specified value rather than the name given by `uname -n`.

An example might help to explain how the entries in the Permissions file work. Suppose that the system named North in the sample network has the following Permissions file.

```
LOGNAME=uwest  MACHINE=west  READ=/  WRITE=/       \
  COMMANDS=uucp:mail  NOREAD=/usr/private   \
  NOWRITE=/usr/private  SENDFILES=yes  REQUEST=yes \
  VALIDATE=west

LOGNAME=nuucp  MACHINE=OTHER  REQUEST=yes  \
  SENDFILES=call
```

The first entry in this file specifies the options that are in effect when a remote system logs in as `uwest`. Because of the `VALIDATE=west` option, the only remote system that can use this user ID is West. When West calls North and logs in as *uwest*, it can read from and write to all directories except the ones starting with `/usr/private` and can execute the commands `uucp` and `mail` on North's system. This entry also includes the `MACHINE=west` option, meaning the options given also apply when North has called West and control has been transferred to North's `uucico` utility. Because `REQUEST=yes` and `SENDFILES=yes`, either system can request or send working files.

The second entry specifies the options in effect when a remote system logs in with the NUUCP user ID. Because `MACHINE=OTHER`, these options will also apply when North has called any remote system except west (which has its own entry) and control has been transferred to North's `uucico`. Files can only be read from or written to the `/usr/spool/uucppublic` directory (no READ or WRITE options to change the default). Either system can request files from the other, but working files are only transferred from north when it calls the remote system.

Tip: When a z/OS system or uucp login is specified, the name must be specified in uppercase.

How uucico uses configuration files

When `uucico` is invoked, it searches the information provided by the `Systems` file (and compiled into the configuration file) for the remote system indicated on its command line. If the `sched` field of the matching entry indicates that it is valid to contact the remote system at that time, `uucico` then checks to see if a connection with the remote system is already in progress (a lock exists in `/usr/spool/locks` for the system or system IP address). If a connection is already in progress, `uucico` will not initiate another connection at this time. Otherwise, `uucico` will attempt to open a TCP/IP connection using the remote system's IP address. If successful, it uses the contents of the `chat_script` field to complete the connection.

Compile the configuration files

In z/OS UNIX, UUCP does not use the configuration files directly. Instead, it uses a special configuration file named `config` which is created when the administrator runs the `uucc` utility to compile the configuration files.

After you set up the configuration files, change directories and then issue:

```
cd /var/uucp
/usr/lib/uucp/uucc
```

A compiled configuration file is created in the `/var/uucp` directory using the `uucc` command from `/usr/lib/uucp`. It contains all of the information specified in the individual configuration files.

The configuration file must be owned by the uucp user ID. If you run `uucp` from any other user ID, you must change the owner of the configuration file from that user ID to uucp.

Do not edit the configuration file directly. If you need to change your configuration, first edit the configuration files and then run `uucc` again.

Create working directories for the local and remote systems

UUCP requires a working directory in `/usr/spool/uucp` for the local system and for each system defined in the Systems file. Each directory must be owned by `uucp` and have *uucpg* as its group ID. If you

create the directories with the *uucp* user ID, this will happen automatically. Otherwise, you will need to **chown** these directories from a superuser user ID.

- Create a working directory for the local system. Enter:

```
mkdir -m 774 /usr/spool/uucp/$(uuname -l)
```

($(uuname -l) will be replaced with the name of your system). If the directory is not owned by uucp and uucpg, enter:

```
chown uucp:uucpg /usr/spool/uucp/$(uuname -l)
```

- Create working directories for remote systems. (If you are setting up your uucp environment for the first time, see the Tip at the end of this step.) For each remote system, enter:

```
mkdir -m 774 /usr/spool/uucp/system
```

where *system* is the name of the remote system.

If the directories are not owned by uucp and uucpg, enter:

```
chown uucp:uucpg /usr/spool/uucp/system
```

where *system* is the name of the remote system.

If you are setting up your UUCP environment for the first time,

```
cd /usr/spool/uucp
mkdir -m 774 $(uuname)
chown uucp:uucpg $(uuname)                # if necessary.
```

$(uuname) will be replaced with a list of all systems defined in the Systems file

Schedule periodic UUCP transfers with cron

UUCP provides two daemons (uucico and uuxqt) which establish communication sessions, transfer data, and execute commands according to the requests scheduled byuucp (for file exchange) and uux (for command execution). uux will invoke uucico unless the command is a local one, in which case it will invokeuuxqt to process the local command immediately. While you can invoke these daemons interactively as the need arises, this becomes inconvenient if many users become dependent on UUCP's capabilities, or if the system must receive data from other UUCP systems according to some regular schedule.

You might need to use **cron** for two reasons:

1. To process requests that were left on the request queue when uucico could not connect.
2. To get files that are waiting to be received from other systems.

The **cron** facility can be used to run the UUCP daemons according to a fixed schedule such as:

- Monday through Friday at 7:30 PM.
- Each day at 8:00 AM. and noon.
- Every 15 minutes starting at midnight.

It is also used to initiate work on behalf of others at predefined times. A crontab file defines the work to be done for a user and the schedule for running it. Use the crontab command to create the crontab file. After a user creates a crontab file (assuming that the **cron** facility has been configured by the administrator), **cron** initiates the work according the specified schedule.

If the UUCP daemons are running from **cron** and encounter an error, they send mail to the user who ran the uucp or uux command that the daemons are processing. The daemons log their status and errors in two files: /usr/spool/uucp/ERRLOG is used to log errors and /usr/spool/uucp/LOGFILE is used to log non-error status. Check those files if the daemons uucico or uuxqt do not seem to be running correctly.

Creating a crontab entry

You can use **cron** to run the UUCP daemons on a fixed schedule. For example, if you want to run the UUCP daemons every Monday through Friday at 7:30 p.m., you would:

1. Log on as UUCP or su to the UUCP user ID
2. Enter the following echo command to create a working copy of the desired crontab entry:

   ```
   echo '30 19 * * 1-5 /usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;' >tfile
   ```

   where

   - 0 means zero minutes
   - 19:30 means 7:30 p.m.
   - * means no selected day of the month
   - * means no selected month of the year
   - 1-5 means Monday through Friday
   - `/usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;` are the commands to be run
   - `>tfile` directs the output to a temporary file
3. Enter the `crontab` command to activate your request.

   ```
   crontab tfile
   ```
4. To display your current crontab entries, enter the following command:

   ```
   crontab -l
   ```

Do not issue the `crontab` command without any options. If you do, the system will erase your current crontab entries and accept new crontab entries from the terminal. If you accidentally enter `crontab` without any options, end it with the INTERRUPT key, which by default is <`Crtl-C`>.

Example of schedules

Here are some examples of other schedules and their crontab entries:

1. Every day at 8:00 a.m. and noon:

   ```
   0 8,12 * * * /usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;
   ```

   where:

   - `0` specifies what minute
   - 8 and 12 specify 8 a.m. and noon
   - * is every day of the month
   - * is every day of the year
   - * is every day of the week
2. Every fifteen minutes starting at midnight:

   ```
   0,15,30,45 * * * * /usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;
   ```

   where

   - `0, 25, 30,` 45 indicates every 15 minutes
   - * specifies every hour of the day
   - * specifies every day of the month
   - * specifies every month of the year
   - * specifies every day of the week

There are many other scheduling possibilities. For more information, see crontab - Schedule regular background jobs in *z/OS UNIX System Services Command Reference*.

Controlling calls to each system

By default, **uucico** attempts connection to every system listed in the Systems file. To reduce these attempts, you can code acceptable callout times for each system in the Systems file. In addition, each system will be contacted even if no data transfers or remote command executions have been requested on the local system. (This is to receive data transfers or local command executions that were requested on the remote system.)

You can specify different crontab entries for different systems. Each of these crontab entries will specify a command of the form

```
uucico -r1 -s site
```

where *site* is the name of the remote site to be called.

Example

This sample `crontab` entry calls the system named North every hour on the hour:

```
0 * * * * /usr/lib/uucp/uucico -r1 -s north
```

Example

This sample `crontab` entry calls the system named East at 12:00 noon and 7:00 p.m. each day from Monday to Friday:

```
0 12,19 * * 1-5 /usr/lib/uucp/uucico -r1 -s east
```

Testing the connection

Once your local system is configured and a remote system is configured, you must test the connection.

1. Change directories to the public UUCP directory.

   ```
   cd /usr/spool/uucppublic
   ```

2. Queue a file request with this command:

   ```
   uucp -mr testfile remote!&tilde;/
   ```

   where *testfile* is the name of a file in the public UUCP directory and *remote* is the name of the remote system.

3. Force a connection:

   a. If the remote system calls your system, have the remote system administrator attempt a connection.

   b. If your system calls the remote system, force a connection with this command:

   ```
   /usr/lib/uucp/uucico -f -r 1 -s remote -x 5
   ```

   where *remote* is the name of the remote system.

4. If everything has been configured correctly, the file is transferred and you can read the mail with `mailx`.

Checking the configuration for connections

If there are problems, check the configuration for this connection. In solving UUCP problems, try to determine how far the connection proceeds before failing. Every UUCP connection goes through these stages:

1. One system establishes a TCP/IP connection with another.
2. The contacted system sends a login prompt and the calling system logs in.
3. The systems negotiate protocols.
4. Files are exchanged.
5. The calling system hangs up.

Try to determine at what stage your connection breaks down. Examine the file `/usr/spool/uucp/LOGFILE` for clues.

Contacting the remote site

If `uucp` cannot establish a connection with the remote system, one of the IP address for the remote system might be wrong or the network path between your system and the remote system might be down. You should be able to ping the remote system to confirm this.

Example

If the address of the remote system west is `west.ibm.com`, enter:

```
tso 'ping west.ibm.com'
```

The following display indicates a successful connection:

```
PING V3R1: PINGING HOST west.ibm.com  (120.40.41.3).
USE ATTN TO INTERRUPT.
PING: PING #1 RESPONSE TOOK 0.043 SECONDS. SUCCESSES SO FAR 1.
```

Calling system login

If the call is made but is not answered, then the login sequence might be at fault. Check `/usr/spool/uucp/LOGFILE` for the record of the sequence exchanged.

- Check the login send/expect sequences specified in the Systems file.
- If the log shows failed logins and the message `you are unknown to me`, confirm that both systems have the correct login name and password and their configurations are set up correctly.

Maintaining UUCP

To maintain UUCP, you need to:

- Read and remove log files periodically. Check `/usr/spool/uucp/LOGFILE` for log files that can be removed..
- Use the **uustat** command to check the status file to ensure that files are transferring to remote systems. Periodically update the configuration files to reflect changes in your system.

Cleaning up UUCP files

Some UUCP files will reside on your UUCP system after it is configured. Here are some pointers on how to clean up old files and make sure that all necessary files are present.

The spool directory

The spool directory holds all work requests and all log files for UUCP. File transfers can be requested by the `uucp` and `uux`.

For each remote system specified in the Systems configuration file, there is a subdirectory in the `/usr/spool/uucp` directory named for the system (for example, the subdirectory for a remote system named South is `/usr/spool/uucp/south`). This subdirectory contains:

- File transfer requests for a remote system.
- Data files for file transfer requests for a remote system.

UUCP data files are created from:

- `uucp` with the `-C` option.
- `uux` requests whose arguments are files that are not on the system running the requested command.
- Execute files, commands that a user on the remote system has requested be run on your system. When you run **uuxqt**, it looks for files in those directories and runs the commands indicated (if all of the permissions are correct).
- The `.Xqtdir` subdirectory, which acts as a working directory when **uuxqt** runs remote commands. After finishing a command, **uuxqt** removes working files from this directory.

Log files, lock files, status files, and working files

UUCP creates system files in the spool directory or in its *system* subdirectories. The following types of system files are created:

Log files
Records of events such as file transfers, deletions, attempts to connect with other systems, and system errors. The spool directory contains the following log file: `/usr/spool/uucp/LOGFILE`. It contains the record of when jobs were queued and executed.

This log file can grow indefinitely. You should edit or delete it on a regular basis.

Lock files
Temporary files created to prevent two programs writing to a file or device simultaneously. These files will have a name of `LCK..`*`site`* and `LCK..`*`site_address`*.

Status files
Records of the most recent unsuccessful attempt to contact a remote system. There is one status file for each remote system you contact; status files are named `/usr/spool/uucp/.Status/`*site*, where *site* is the remote site's name.

You can use the `uustat -q` command to view the contents of the status file.

A status file is only created if the last attempt to contact a system was unsuccessful. The status file is not required for UUCP to attempt another call.

Working files
Command, data, and execute files for the UUCP file transfer programs, stored in the appropriate subdirectory for the system.

Displaying information about recorded UUCP events

You can use the `uulog` command to display information about recorded UUCP events, such as file transfers and remote command execution.

Notifying remote systems about password changes

When you change the password of a user ID (for example, NUUCP) that is used for a remote system to login to the local system, you must notify each remote system to update its Systems file chat script with the new password.

Chapter 11. Converting files between code pages

z/OS is an EBCDIC platform. It has devices that are configured for EBCDIC; it also has programs that are compiled to handle the EBCDIC encoding of characters. If you implement Enhanced ASCII or exploit Unicode Services, the basic EBCDIC nature of a z/OS platform remains. For example, the z/OS shell and utilities continue to be EBCDIC programs. However, if C programs were compiled as ASCII, the EBCDIC nature can be partially hidden.

Before Enhanced ASCII and Unicode Services were available, you could use **iconv** to convert characters from one code page set to another. The converted text is written to standard input (stdout).

C programs that have been compiled as ASCII use ASCII locales. These ASCII locales are produced by using the `-A` option of `localedef`.

List of subtasks

This topic covers the following subtasks:

Subtask	Associated procedure
Setting up enhanced ASCII	“Setting up Enhanced ASCII” on page 250
Setting up Unicode Services	“Steps for setting up Unicode Services” on page 252

Enhanced ASCII

Enhanced ASCII introduces automatic conversion, which, in some cases, is an alternative to **iconv**. The enhanced ASCII functionality makes it easier to port internationalized applications developed on ASCII platforms, or for them, to z/OS platforms by providing conversion from ASCII to EBCDIC, and from EBCDIC to ASCII. Enhanced ASCII also provides support for file tagging. File tags are a way to identify the code set of text data within files and are used during automatic conversion.

Enhanced ASCII provides limited conversion of ASCII to EBCDIC, and EBCDIC to ASCII. The character set or alphabet that is associated with any locale consists of the following character subsets:

- A common, XPG4-defined subset of characters such as POSIX portable characters.
- A unique, locale-specific subset of characters such as globalization characters.

Restriction: The conversion only applies to the portable subset of characters that are associated with a locale. Only the EBCDIC IBM-1047 encoding of portable characters is supported.

You might encounter unexpected results in the following situations:

- If Enhanced ASCII applications run in locales that contain non-Latin Alphabet Number 1 characters, C-RTL functions might copy some of the locale's non-Latin 1 characters into buffers that the application is writing to stdout or another file. The non-Latin Alphabet Number 1 characters would then cause problems during automatic conversion.
- Language Environment applications might select non-English message files. If the NATLANG runtime option is not ENU or UEN, then conversion does not take place. The messages are presented to the file system write routine in EBCDIC, before any automatic conversion takes place. If the automatic conversion is to EBCDIC, then there will be a problem because EBCDIC cannot be converted to EBCDIC.

A subset of C headers and functions is provided in ASCII. For a list of all C/C++ runtime functions that support Enhanced ASCII, see Enhanced ASCII support in *z/OS XL C/C++ Runtime Library Reference*.

The only way to get to the ASCII version of functions and the external variables *environ* and *tzname* is to use the appropriate IBM header files. ASCII applications may read, but not update, environment variables using the *environ* external variable. Updates to the environment variables that use *environ* in an ASCII

application causes unpredictable results and might result in an abend. Language Environment maintains two equivalent arrays of environment variables when running an ASCII application, one with EBCDIC encodings and the other with ASCII encodings. All ASCII compile units that use the *environ* external variable must include <stdlib.h> so that *environ* can be mapped to access the ASCII encoded environment strings. If <stdlib.h> is not included, *environ*will refer to the EBCDIC representation of the environment variable strings.

To execute ASCII shell scripts and REXX execs, use spawn (BPX1SPN).

Setting up Enhanced ASCII

Limit the enabling of automatic conversion to the smallest environment possible. One way to accomplish this is by using the _BPXK_AUTOCVT environment variable, optionally with the FILETAG runtime option.

It is important to understand that file tagging and enabling automatic conversion are independent operations. Because you can tag files without enabling automatic conversion, and vice versa, it is possible to have many tagged files without any conversion occurring. However, if you enable automatic conversion for the entire system by using the AUTOCVT statement in BPXPRMxx, each tagged file becomes subject to conversion by any program that reads from or writes to those tagged files. Thus, programs may be processing converted data even though they do not support it. (An example would be an EBCDIC program that expects to read an ASCII file as ASCII data.) For those reasons, it is a good idea to limit the enabling of automatic conversion to the smallest environment possible using _BPXK_AUTOCVT and, if applicable for the C run time environment, the FILETAG runtime option.

Perform the following steps to set up Enhanced ASCII.

1. Set up Enhanced ASCII. Base your choice on your particular situation.

If this situation exists . . .	**Then use . . .**
The application is written in C/C++.	The FILETAG runtime option with the _BPXK_AUTOCVT environment variable.
The application is run in the z/OS UNIX shell or BPXBATCH.	The _BPXK_AUTOCVT environment variable.
You are enabling automatic conversion for the z/OS UNIX environment.	AUTOCVT(ON) statement in the BPXPRMxx parmlib member. You can use the SETOMVS and SET OMVS operator commands to turn AUTOCVT on or off.

2. Assign the appropriate file tag for each file that is to be converted. Base your choice on your particular situation.

If you choose this method . . .	**Then this happens . . .**
Issuing the `chtag` command.	Files are permanently tagged.
Mounting a file system with the TAG parameter.	Files are temporarily tagged. All untagged files in the file system that is being mounted are implicitly tagged. When the file system is unmounted, the tags are lost.
Issuing the F_SETTAG subcommand of the BPX1FCT (fcntl) callable service from a program.	Files are either temporarily or permanently tagged, depending on the input parameters. For more information about the subcommand, see fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

If you choose this method . . .	Then this happens . . .
Issuing BPX1CHR (chattr) callable service from a program.	Files are permanently tagged. For more information about BPX1CHR, see chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.
Issuing fopen() or popen() with the 'text' option and the C-RTL FILETAG(,AUTOTAG) runtime option.	New or empty files are automatically tagged at first write(). Programs that use this form of opening a file are already set up for tagging, and require the least effort to set up automatic conversion.

3. Assign a coded character set identifier (CCSID) to each program or thread in the shell. By default, the initial CCSID for every thread is IBM-1047 (EBCDIC).
   - For entire programs written in C/C++, use the ASCII compiler to change it to 819 (ISO8859-1 ASCII).
   - For C/C++ threads, use the F_CONTROL_CVT subcommand of fcntl().
   - For Assembler programs and threads, use the F_CONTROL_CVT subcommand of the BPX1FCT callable service. F_CONTROL_CVT sets the CCSID of the program associated with each opened file. (That is, the program CCSID can be different depending on which file is chosen.)
   - Use the mapping macro BPXYTHLI to set field Thliccsid. IBM no longer recommends this method.

When you are done, you have set up Enhanced ASCII.

Using Unicode Services in a z/OS UNIX environment

z/OS UNIX exploitation of Unicode Services is functionally similar to that provided for Enhanced ASCII, which is described in "Enhanced ASCII" on page 249. The basic EBCDIC nature of the z/OS platform remains. Likewise, programs cannot alter their EBCDIC nature as compiled units, except for C programs, which can be compiled as ASCII. Locale restrictions that apply to Enhanced ASCII functions apply to Unicode Services functions as well.

Files that are tagged can be converted between any CCSID of the program or user and the CCSID of the file, if Unicode Services supports that conversion. Unlike Enhanced ASCII, which affects conversion of regular file, pipes, and character special files, an environment enabled for Unicode Services environment affects regular files and pipes only. No character special support beyond that provided for Enhanced ASCII is included.

For more information about Unicode Services, see *z/OS Unicode Services User's Guide and Reference*.

Considerations beyond that of Enhanced ASCII

Because POSIX standards apply, z/OS UNIX still treats text that is to be (or has been) converted as a series of bytes. Read and write request sizes, return values, file offsets, file sizes, and so on, remain byte-designated values. However, because one byte does not always equal one character, complications can result when the LFS is requested to perform text conversion. For example, for a write operation, a program might receive a return value equal to the number of bytes that the program requested, but z/OS UNIX might write a different amount due to converting the data. Typically, a program would not be aware of this action, but file sizes increase or decrease differently due to conversion. A large write operation might fail because the maximum number of bytes that are allowed (2 G default for a regular file) was exceeded, even though the program specified less than 2 G.

Incomplete characters at the end of an I/O stream might cause problems. This situation occurs, for example, when z/OS UNIX receives data that does not end on a character boundary. z/OS UNIX tries to resolve this problem by caching the end of this data for the next I/O operation. But unconverted data will

not be hardened. Consequently, an fsync operation will not cause a partial character to be written to the file. Likewise, closing a file with a partial character held by z/OS UNIX causes that partial character to be lost. Partial characters occur only when multibyte characters sets are being converted.

Additionally, when certain multibyte character sets are being converted, a lseek operation can cause the file offset to jump to a location in the file that is not on a character boundary or that contains a different character set. These jumps can cause a subsequent read or write operation to fail with an I/O error or a conversion error. Sequential reading and writing is the preferred I/O method to use when different character sets exist in the file.

If the CCSID has multibyte character sets, partial characters may occur and Unicode Services might fail the conversion with the EIO return code. In this case, adjust the _BPXK_UNICODE_SUB, _BPXK_UNICODE_TECHNIQUE, or _BPXK_UNICODE_MAL environment variable for Unicode Services to take the appropriate conversion action. You can also turn off automatic conversion and use character-based functions such as **iconv** instead.

Steps for setting up Unicode Services

Setting up for Unicode Services conversion is similar to setting up for Enhanced ASCII with the following changes:

- _BPXK_AUTOCVT can be set to ALL, which enables a Unicode Services conversion environment for the program or user.
- AUTOCVT(ALL) can be specified in the BPXPRMxx parmlib member, which enables a Unicode Services conversion program for all programs and users. The SETOMVS or SET OMVS operator commands can turn AUTOCVT to ALL.
- Instead of setting the **ThliCcsid** field the environment variable _BPXK_PCCSID should be set. For a description of the _BPXK_PCCSID environment variable, see "_BPXK environment variables" on page 405.

Perform the following steps to set up Enhanced ASCII.

1. Set up Unicode Services. Base your choice on your particular situation.

If this situation exists . . .	Then use . . .
The application wants to exploit Unicode Services when enabling automatic conversion.	The _BPXK_AUTOCVT environment variable can be set to ALL or fcntl(), or BPX1FCT can be called to enable to conversion. For the XL C/C++ runtime environment, refer to *z/OS XL C/C++ Programming Guide*.
The application is run in the z/OS UNIX shell or BPXBATCH.	The _BPXK_AUTOCVT environment variable. _BPXK_AUTOCVT can be set to ALL, which enables a Unicode Services conversion environment for the program or user.
You are enabling automatic conversion for the z/OS UNIX environment.	AUTOCVT(ALL) in BPXPRMxx, which enables Unicode Services conversion environment for all programs and users. **Tip:** Use the SETOMVS and SET OMVS operator commands to turn AUTOCVT on or off.

2. Assign the appropriate file tag for each file that is to be converted. Base your choice on your particular situation.

If you choose this method . . .	Then this happens . . .
Issuing the `chtag` command.	Files are permanently tagged.

If you choose this method . . .	Then this happens . . .
Mounting a file system with the TAG parameter.	Files are temporarily tagged. All untagged files in the file system that is being mounted are implicitly tagged. When the file system is unmounted, the tags are lost.
Issuing the F_SETTAG subcommand of the BPX1FCT (fcntl) callable service from a program.	Files are either temporarily or permanently tagged, depending on the input parameters. For more information about BPX1FCT, see fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.
Issuing BPX1CHR (chattr) callable service from a program.	Files are permanently tagged. For more information about BPX1CHR, see chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

_____

3. Assign a coded character set identifier (CCSID) to each program or thread in the shell. By default, the initial CCSID for every thread is IBM-1047 (EBCDIC).
   - For entire programs written in C/C++, use the ASCII compiler to change it to 819 (ISO8859-1 ASCII).
   - For C/C++ threads, use the F_CONTROL_CVT subcommand of fcntl().
   - For Assembler programs and threads, use the F_CONTROL_CVT subcommand of the BPX1FCT callable service. F_CONTROL_CVT sets the CCSID of the program associated with each opened file. (That is, the program CCSID can be different depending on which file is chosen.)
   - Set environment variable _BPXK_PCCSID.

When you are done, you have set up Unicode Services.

Chapter 12. Managing operations

The z/OS UNIX element is designed to be continually available. This topic discusses tasks that are done by operators.

List of subtasks

Subtasks	Associated procedure
Ending a specified process	"Steps for ending a specified process" on page 255
Shutting down z/OS UNIX	"Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=..." on page 258 "Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN" on page 262
Doing partial shutdowns for JES maintenance	"Steps for partial shutdowns for JES2 maintenance" on page 260
Dynamically adding FILESYSTYPE parameters in BPXPRMxx	"Steps for activating the zFS file system for the first time" on page 269 "Steps for activating a single sockets file system for the first time" on page 270 "Steps for activating a multiple socket file system for the first time with Common INET" on page 271 "Steps for increasing the MAXSOCKETS value" on page 271 "Steps for adding another sockets file system to an existing CINET configuration " on page 272

If you require a high level of security in your z/OS system and do not want superusers to have access to z/OS resources such as SYS1.PROCLIB, read the following topics:

- "Comparing UNIX security and z/OS UNIX security" on page 311.
- "Establishing the correct level of security for daemons" on page 313.

Steps for ending a specified process

A *process* is the execution of a program. MVS calls the basic unit of execution a job or a task; in UNIX, it's called a process. You can find out whether a process is active, and you can end it.

Before you begin: You need to know which processes you want to end and whether they are active.

Use the DISPLAY OMVS operator command or the **ps** command to display all active processes.

1. To list the address space identifiers for processes, issue:

```
DISPLAY OMVS,A=ALL
```

2. To display a particular ASID, where *asid* is a specified ASID:

```
DISPLAY OMVS,A=asid
```

3. To display information about all accessible processes, providing that you have the appropriate privileges:

```
ps -elf
```

Perform the following steps to end a specified process, where *pppp* is the process identifier (pid).

1. Send a SIGTERM signal by using the shell **kill** command or use the TERM parameter of the MODIFY operator command. For example:

   a. `kill -s term pppp`

   b. `F BPXOINIT,TERM=pppp`

   If the process is not ended, then go to Step "2" on page 256. Otherwise, you have canceled the process and you are finished.

   ———————————————————————————————————————

2. Send a SIGKILL signal by using the shell **kill** command or use the FORCE parameter of the MODIFY operator command. For example:

   a. `kill -s kill pppp`

   b. `F BPXOINIT,FORCE=pppp`

   If the process is still not ended, then go to Step "3" on page 256. Otherwise, you have canceled the process and you are all done.

   ———————————————————————————————————————

3. Send a stronger SIGKILL signal by using the shell **kill** command or use the SUPERKILL parameter of the MODIFY operator command. For example:

   - `kill -K pppp`
   - `F BPXOINIT,SUPERKILL=pppp`

   If the process is still not ended, then go to Step "4" on page 256. Otherwise, you have canceled the process and you are finished.

   ———————————————————————————————————————

4. If the previous steps did not end the process, then use the CANCEL command. Issue:

   ```
   CANCEL jobname,a=asid
   ```

   The following example shows how to obtain the ASIDs for a user with the TSO/E user ID JOE and then cancel the user's process that is running the **sleep 6000** 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
```

_____

When you are done, you have ended the specified process.

Ending threads

A *thread* is a stream of computer instructions that is in control of a process. Several threads can run concurrently, performing different jobs. An operator can use the MODIFY command to end a thread without disrupting the entire process. The syntax is:

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

where

- `pid` indicates the process identifier (PID) of the thread to be ended. 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 ended. 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 ended.
- FORCE= indicates the signal interface routine will not be allowed to receive control before the thread is ended.

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

You will typically want to end 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, use the MODIFY operator command to terminate the thread without ending the entire process. Use the TERM keyword first. If that does not succeed, then use FORCE. For example:

- To allow the signal interface routine to receive control before the thread is ended:

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

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

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

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

Planned shutdowns using F BPXOINIT,SHUTDOWN=...

When you are doing a planned shutdown of z/OS UNIX, follow the instructions in "Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=..." on page 258. As part of a planned shutdown, you need to prepare the file systems before shutting down z/OS UNIX by issuing one of these commands:

```
F BPXOINIT,SHUTDOWN=FILEOWNER
F BPXOINIT,SHUTDOWN=FILESYS
```

Issuing one of these commands synchronizes data to the file systems and possibly unmounts or moves ownership of the file systems. If you use SHUTDOWN=FILEOWNER, the system is disabled as a future file system owner via move or recovery operations until z/OS UNIX has been restarted.

Restriction: SHUTDOWN=FILEOWNER is valid only in a shared file system configuration.

If you get message BPXM048I saying that the file system shutdown was incomplete, a local mount might have been performed while the shutdown was in progress. To identify the file systems that were not moved or unmounted, issue D OMVS,F,O on the source system to see which file systems are still owned by this system. You can try to move individual file systems by issuing the following operator command for each file system in question:

```
SETOMVS FILESYS,FILESYSTEM=xxxxxxxxx,SYSNAME=yyyyyyyy
```

If a move fails, you will see message BPXO037E.

Automounted file systems are not mounted during the processing of F BPXOINIT,SHUTDOWN=FILEOWNER, or after it completes in order to provide the ability to handle unexpected mounts that occur when a file system is shut down.

In a shared file system configuration, the resulting system actions are more complex, because they might involve the movement of file system ownership between systems in the shared file system. For more information about the system actions that might occur in a shared file system, see "Implications of shared file systems during system failures and recovery" on page 184.

To shut down the system as part of JES maintenance without reIPLing the system, see "Partial shutdowns for JES2 maintenance" on page 260.

Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=...

About this task

You will shut down z/OS UNIX using the F BPXOINIT,SHUTDOWN=... system command.

Before you begin, you need to notify users that the system is being shut down and ask them to log off. If you do not shut down and quiesce the UNIX workload, these critical system functions might be ended abnormally during the shutdown, which might cause several failures on the system. As a result, the system might not be shut down successfully.

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 shell users who are logged on. For example:

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

Perform the following steps to shut down z/OS UNIX using F BPXOINIT,SHUTDOWN.

Procedure

1. Prevent new TSO/E logons and shut down other z/OS subsystems (such as CICS® and IMS), following your typical procedures.

   _____

2. Shut down all JES initiators.

   _____

3. Move or unmount all of the NFS file systems by issuing the following command:

   ```
   F OMVS,STOPPFS=NFS
   ```

   _____

4. Use normal shutdown procedures to end 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 ending.

   _____

5. End running daemons such as `inetd`. Then end any remaining processes.

   To obtain a list of daemons that are running, issue:

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

   OMVSKERN is the user ID that is used for the kernel and daemons.

   To display all processes (most daemons have recognizable names), issue:

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

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

   _____

6. Move or unmount all file systems (including the root file system). Issue:

   ```
   F BPXOINIT,SHUTDOWN=FILEOWNER
   ```

   or

   ```
   F BPXOINIT,SHUTDOWN=FILESYS
   ```

   _____

7. Take down JES. At this point, there might 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, but you can end them by issuing:

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

   _____

Results

When you are done, you have ended all of the processes. 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, log on 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 topic explains how you can terminate all of the forked processes without having to reIPL 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 end 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.

Steps for partial shutdowns for JES2 maintenance

Before you begin, you need to notify users that the system is being shut down and ask them to log off. If you do not shut down and quiesce the UNIX workload, these critical system functions might be ended abnormally during the shutdown, which might cause several failures on the system. As a result, the system might not be shut down successfully.

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 shell users that are logged on. For example:

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

Perform the following steps to accomplish a partial shutdown.

1. Shut down z/OS UNIX.

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

   All forked and non-local spawned address spaces on the system are ended. If the operator receives a success message, the shutdown can be continued.

   failure message means that some forked processes or non-local spawned address spaces could not be ended. To find these processes, issue:

   ```
   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.

   ——

2. Stop the file system colonies that were started under JES (those without SUB=MSTR specified when they were defined). Use normal shutdown procedures to close all file system address spaces such as Network File System Client (NFSC).

For NFSC use the F OMVS,STOPPFS=NFS command to bring down NFSC.

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

When you are done, you have partially shut down z/OS UNIX. New fork and spawn activity cannot be done; however, it is still possible for batch jobs and TSO users to use z/OS UNIX services. Now you can do whatever corrective or maintenance actions that were needed for JES2, such as restarting it.

To restart z/OS UNIX:

1. Issue the MODIFY (F) command.

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

2. Restart the file system address spaces.

   For NFSC, respond to the operator message BPXF032D issued when you previously stopped NFSC using F OMVS,STOPPFS=NFS command. Then reissue all the mounts if required.

   For DFSCM, respond to the operator message BPXF032D.

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

Planned shutdowns using F OMVS,SHUTDOWN

"Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN" on page 262 describes how to use the F OMVS,SHUTDOWN operator command for a planned shutdown and reIPL. Consider using it if you plan to recustomize and reinitialize the z/OS UNIX environment without reIPLing. Using F OMVS,SHUTDOWN along with F OMVS,RESTART might allow you to avoid a system outage by providing the ability to shut down and then reinitialize the z/OS UNIX environment without the need for a reIPL.

F OMVS,SHUTDOWN is an alternative to F BPXOINIT,SHUTDOWN=... commands for synchronizing data to the file systems and unmounting or moving file system ownership before a planned shutdown.

Sometimes F OMVS,SHUTDOWN cannot shut down z/OS UNIX completely and you must do a reIPL in order to correct the condition that is requiring the shutdown. With F OMVS,SHUTDOWN and F OMVS,RESTART, some reconfiguration tasks can be accomplished that otherwise would have required a reIPL. These tasks include the following steps:

- Reconfiguring a system to go from a non-shared file system to a shared file system.
- Completely applying a new file system file structure.

However, there are some tasks that you cannot accomplish using these commands, as follows:

- Installing maintenance to the z/OS UNIX component.
- Resolving severe system outages.

Follow the recommended pre-shutdown procedure described in "Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN" on page 262 when using F,OMVS,SHUTDOWN. If you do not, the risk of having to do a reIPL is far greater.

If you want to shut down the system as part of JES2 maintenance and do not want to reIPL the system, issue F BPXOINIT,SHUTDOWN=FORKS as described in "Partial shutdowns for JES2 maintenance" on page 260.

What F OMVS,SHUTDOWN does

Use the F OMVS,SHUTDOWN operator command when you want to do a planned shutdown, and might or might not be reIPLing the system. You will be shutting down the entire z/OS UNIX system and all processes.

Only eligible running processes are shut down. Some processes might not be shut down because they have registered as a permanent process. Additionally, some applications might register to block

a shutdown, which delays the shutdown request until the blockers end or unblock. Also, an application exit can be set up to be given control when a shutdown request is initiated in order to allow specific shutdown actions to be taken. This might include initiating the shutdown of the application or sending messages that indicate the specific steps that are required to shut down the application.

If any blocking jobs or processes are active when a shutdown request is initiated, the shutdown is delayed until all blocking jobs or processes either unblock or end. If the delay exceeds a certain time interval, you will receive messages telling you that the shutdown is delayed and which jobs are delaying the shutdown. At this point, you can either attempt to end the jobs that are identified as blocking shutdown or issue F OMVS,RESTART to restart the z/OS UNIX environment, which will cause the shutdown request to be ended.

Successful shutdowns

The shutdown succeeds only if all non-permanent z/OS UNIX processes end, all permanent processes are successfully checkpointed, and if all physical file systems are successfully quiesced. Otherwise, the shutdown request will fail. Nonpermanent processes within jobs that are not prepared for shutdown cause the shutdown request to fail. These jobs are identified in messages so that you can force these jobs to end. At this point, because most processes have been ended, you should force the hung jobs to end and then try the shutdown again. Because some jobs might have ended abnormally, JES spool resources might have accumulated for these jobs; you will have to purge them using commands such as $POJOBQ,READY.

At each phase of shutdown, it is possible that there could be a stall where no shutdown activity is occurring. That situation could cause the shutdown to hang. If such a situation is detected, the shutdown will wait approximately six minutes for the stall to resolve itself. If the stall does not resolve itself by then, the shutdown request will fail.

Because some resources are tied to components outside of the scope of the kernel (shared memory, mmap, shared libraries, for example), you must end any application that is using any of these resources before z/OS UNIX can be ended, including applications that are registered as permanent.

Because the F OMVS,SHUTDOWN support encompasses the existing support in the F BPXOINIT,SHUTDOWN= command, you do not need to issue F BPXOINIT,SHUTDOWN before using F OMVS,SHUTDOWN. If F OMVS,SHUTDOWN fails, z/OS UNIX services are reenabled whether or not a F BPXOINIT,SHUTDOWN= was done prior to the F OMVS,SHUTDOWN command. An F BPXOINIT command of any kind issued when OMVS is shut down is ignored.

CAUTION: Use F OMVS,SHUTDOWN carefully because this method will take down other system address spaces. As a result, some system-wide resources might not be completely cleaned up during a shutdown and restart. Do not use this command to shut down and restart the z/OS UNIX environment on a frequent basis. (If you do so, you will eventually have to do a reIPL.) An example of a system-wide resource that can be consumed due to the shutdown are non-reusable ASIDs. If colony address spaces are being used, a non-reusable ASID will be consumed for each colony address space that is shut down. For this reason, installations should plan on increasing the value set for the RSVNONR= parameter in the IEASYSxx parmlib member to account for the consumption of non-reusable ASIDs due to each shut down of OMVS. If this value is not increased, the installation might receive an error message after shutting down the system multiple times.

If a shutdown does not succeed, a critical z/OS UNIX resource might not have been available during the shutdown, due to a prior system problem, such as latch contention. If a resource such as the z/OS UNIX file system MOUNT latch could not be obtained, the shutdown request to likely to stall and then fail.

Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN

Before you begin, you need to notify users that the system is being shut down and ask them to log off. If you do not shut down and quiesce the UNIX workload, these critical system functions might be ended abnormally during the shutdown, which might cause several failures on the system. As a result, the system might not be shut down successfully.

- 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.',NOW
```

- Use the `wall` command to send a similar note to all logged-on shell users. For example:

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

Perform the following steps to shut down z/OS UNIX using F OMVS,SHUTDOWN.

1. Prevent new TSO/E logons.

   _____

2. Quiesce your batch and TSO workloads. Having batch jobs and TSO users running during the shutdown might cause these jobs to experience unexpected signals or abends. Additionally, these jobs and users might end up being hung, waiting for z/OS UNIX services to be restarted, if they first access z/OS UNIX services during a shutdown.

   Quiesce those application and subsystem workloads using z/OS UNIX services in the manner that each application or subsystem recommends. Doing so will allow subsystems such as Db2, CICS and IMS, and applications like SAP, Lotus Domino, Tivoli NetView for z/OS, and WebSphere® to be quiesced in a more controlled manner than this facility will provide.

   You can use the `D OMVS,A=ALL` operator command to determine which applications, if any, require quiescing.

   _____

3. Move or unmount all of the NFS file systems by issuing F OMVS,STOPPFS=NFS. Doing so prevents the NFS file system from losing data.

   _____

4. Terminate all file system address spaces such as TCP/IP and DFSS, using their recommended shutdown methods. If you do not shut them down before issuing F OMVS,SHUTDOWN, these system functions might terminate abnormally when the shutdown takes place. Do not shut down existing PFS colony address spaces such as zFS because they are shut down as part of F OMVS,SHUTDOWN.

   _____

Now you can issue F OMVS,SHUTDOWN.

Note:

1. After an F OMVS,SHUTDOWN request is accepted, jobs that attempt to use z/OS UNIX services for the first time will be delayed until the system is restarted. Terminating signals are sent to jobs that are already connected; these jobs will be ended abruptly.
2. After F OMVS,SHUTDOWN has completed, you can shut down the system completely via an IPL or by powering off.

   You can completely restart and reinitialize the z/OS UNIX environment by issuing F OMVS,RESTART. You can also use it to change the configuration of z/OS UNIX services by specifying a different set of BPXPRMxx members when z/OS UNIX is started.
3. Using F OMVS,SHUTDOWN, the steps for shutting down z/OS UNIX are the same whether or not the system is participating in a shared file system. However, in a shared file system, the resulting system actions are more complex because they might involve the movement of file system ownership between systems in the shared file system. For more information about system actions that might occur in a shared file system, see “Implications of shared file systems during system failures and recovery” on page 184.

Dynamically activating the z/OS UNIX component service items

You can dynamically activate and deactivate some service items (PTFs, ++APARs, ++Usermods) that affect the z/OS UNIX component modules without having to reIPL. This capability is primarily intended to allow an installation to activate corrective service to avoid unplanned reIPLs of your systems. Additionally, this capability can be used to activate temporary code that can be used in gathering additional documentation for a recurring system problem. Although this capability can be used to activate preventive service on an ongoing basis, it is not intended for this purpose.

- F OMVS,ACTIVATE=SERVICE activates the service.
- F OMVS,DEACTIVATE=SERVICE backs off the service.
- D OMVS,ACTIVATE displays the current set of services that were dynamically activated.

Those PTFs that are capable of being activated dynamically will have ++HOLD REASON(DYNACT) included within their PTF. Additionally, any ++USERMOD or ++APAR provided from IBM will have explicit instructions provided by the IBM Service Team indicating whether the ++Usermod or ++APAR can be dynamically activated. Although a service item might be identified as being capable of dynamic activation, the level of a given system might not be current enough to allow the activation of the service item. The ++HOLD REASON(DYNACT) will identify the service level required to activate the PTF. In order to properly activate the PTF, follow the directions in the ++HOLD in the PTF.

Dynamically activating a PTF on top of a superseded ++APAR is not allowed. If the ++APAR was activated dynamically, you can deactivate it and then activate the PTF. Otherwise, the PTF will need to be applied with an IPL.

In order to be prepared to exploit dynamic service activation, you must stay current on z/OS UNIX component maintenance. Staying current makes it more likely that any given service item can be activated dynamically, because the running system will be at a high enough level to accept the service item. On a periodic or as-needed basis, you will have to determine the selected PTFs that you would be interested in activating dynamically for corrective purposes. These would likely be the PTFs that are of highest severity and highest impact related to your workloads.

To ensure that you activate only those service items that are of interest, it is recommended that you additionally install these service items into a separate load library from the LPALIB or LINKLIB libraries that are used for your normal installation process. You can copy (or clone) the SMP/E environment for the currently running z/OS system and then install the applicable PTFs using the cloned SMP/E environment and cloned target libraries. The applicable PTFs are the ones that have been identified with ++HOLD REASON(DYNACT).

Restrictions:

- It is highly recommended that you install service items into a loadlib separate from the LPALIB and LINKLIB libraries that is used for normal installation.
- Once parmlib options SERV_LINKLIB and SERV_LPALIB are set, you cannot unset them, or set them to null. You must clean out old service from the libraries if you continue to use the same libraries for future dynamic activation.
- Both SERV_LINKLIB and SERV_LPALIB must be APF-authorized.

Identifying service items to be activated

Service items are activated from service activation libraries that have been identified via the SERV_LPALIB and SERV_LINKLIB parameters in the BPXPRMxx member. The service activation libraries contain the service items that have already been SMP/E-installed and that you want to activate on the next F OMVS,ACTIVATE=SERVICE command. The libraries enable you to identify for a given activation request the normal LPALIB and LINKLIB target libraries where you install service via SMP/E for future IPLs or a specific library where you have installed a specific fix. For example:

```
SERV_LINKLIB('dsname','volser')
```

The SERV_LINKLIB statement identifies the target service library where the z/OS UNIX modules that are normally loaded from SYS1.LINKLIB into the private area of the OMVS address space are located.

- The *dsname* parameter identifies a 1-to-44 character value that represents a valid data set name for the specified MVS load library. This library must be APF-authorized. The alphabetic characters in the load library name must be uppercase.
- The *volser* parameter identifies a 1-to-6 character value that represents a valid volume serial number for the volume that contains the specified MVS load library. The alphabetic characters in the volume serial number must be uppercase.

Because you can set these new statements via SET OMVS, different target service libraries can be used for any given F OMVS,ACTIVATE=SERVICE command invocation.

Activating service items

After you identify the target service activation libraries via the SERV_LPALIB and SERV_LINKLIB parameters in the BPXPRMxx member, you can then activate the service items.

For most service items that can be activated dynamically, issuing F OMVS,ACTIVATE=SERVICE is sufficient to activate the service item. But for a small number of service items, the procedure to activate a fix or set of fixes might require the dynamic LPA add of BPXINLPA and its ALIASes. This is the case only if the maintenance to be activated affects one of the ALIASes of BPXINLPA. The accompanying documentation provided by IBM for a service item (for example, .++HOLD) will indicate if this extra step is required to activate a service item.

Note the following restrictions:

1. Only those service items found in the target libraries that are identified internally by z/OS UNIX as capable of being dynamically activated are activated. Service items that are not explicitly identified as such cannot be activated.
2. If a fix capable of dynamic activation is found that cannot be activated due to earlier service on the active system or missing parts in the target activation libraries, the activation will fail.

The set of service items to be activated and the amount of ECSA and OMVS address space storage consumed for those service items are indicated in messages displayed by the F OMVS,ACTIVATE=SERVICE command. The issuer of the command is then prompted on whether to proceed with the activation based on this information.

Tip: The activation of a set of service items potentially causes the additional consumption of both ECSA and OMVS address space storage that is permanent regardless of whether the service items are deactivated. A dynamic activation that involves fixes to modules in LPA resident load modules will cause additional consumption of ECSA. Careful consideration should be given to ensure that this additional ECSA storage consumption does not cause a problem for your system.

To activate the component service items, issue:

```
F OMVS,ACTIVATE=SERVICE
```

If a fix capable of dynamic activation is found that cannot be activated due to earlier service found on the active system or missing parts in the target activation libraries, the activation will fail.

Deactivating service items

You can back off a set of dynamically activated service items, if you need to. This might be necessary if, for example, a problem is encountered with a dynamically activated service item or if a particular service item is no longer necessary.

To deactivate the service items, issue:

```
F OMVS,DEACTIVATE=SERVICE
```

Only the service items that were activated when F OMVS,ACTIVATE=SERVICE was last issued are backed off. You will see a list of service items to be deactivated and you will be asked whether the deactivation should proceed.

Displaying activated service items

Use the D OMVS,ACTIVATE=SERVICE command to display all of the service items that were dynamically activated using F OMVS,ACTIVATE=SERVICE. It displays only those service items that are currently active dynamically. Once a fix has been deactivated, it will no longer show up in this command's display output.

Additionally, D OMVS,ACTIVATE=SERVICE reports the library and volume where each set of fixes was activated from and the amount of ECSA and OMVS address space storage that is being consumed for all dynamically activated fixes. The amount of storage consumed will not decrease when a deactivation is done.

Example

To display all the service items that were dynamically activated, issue:

```
D OMVS,ACTIVATE=SERVICE
```

The following is a sample output:

```
BPXO059I 08.51.42 DISPLAY OMVS 284
    OMVS     000E ACTIVE          OMVS=(6D)
                  DYNAMIC SERVICE ACTIVATION REPORT
      SET  #3:                                          LINKLIB=SYS1.DYNLIB.PVT                    VOL=BPXLK1
        LPALIB=SYS1.DYNLIB.LPA                    VOL=BPXLK1
          OA12345  OA23456  OA34567  OA45678  ANLATC1
      SET  #2:
        LINKLIB=SYS1.DYNLIB.PVT                   VOL=BPXLK1      LPALIB=SYS1.DYNLIB.LPA     VOL=BPXLK1
          OA02001  OA02002  OA02003  OA02004  OA02004  OA02005
          OA02007  OA02008  OA02009
      SET  #1:
        LINKLIB=SYS2.DYNLIB.PVT                   VOL=BPXLK1      LPALIB=SYS1.DYNLIB.LPA     VOL=BPXLK1
          OA01001  OA01002  OA01003
       ECSA STORAGE: 1268496       OMVS  STORAGE:  4768248
```

Figure 38. Sample D OMVS,ACTIVATE=SERVICE output

The service items are listed in groups based on when they were activated. All service items activated by a given F OMVS,ACTIVATE=SERVICE command are listed together as one set of activated service items. The most recently activated set of service items is listed first followed in descending order by the next most recent activation set and so on. The most recent set of service items is the only service items that are deactivated if a F OMVS,DEACTIVATE=SERVICE command is issued.

Example

If a F OMVS,DEACTIVATE=SERVICE was done on a system with the service items activated, the following would then show up on a D OMVS,ACTIVATE=SERVICE after the deactivation:

```
BPXO059I 08.58.26 DISPLAY OMVS 296
    OMVS     000E ACTIVE          OMVS=(6D)
                  DYNAMIC SERVICE ACTIVATION REPORT
      SET  #2:
       LINKLIB=SYS1.DYNLIB.PVT                 VOL=BPXLK1             LPALIB=SYS1.DYNLIB.LPA    VOL=BPXLK1
         OA02001  OA02002  OA02003  OA02004  OA02005  OA02006
         OA02007  OA02008  OA02009
      SET  #1:
         LINKLIB=SYS2.DYNLIB.PVT                 VOL=BPXLK1                LPALIB=SYS1.DYNLIB.LPA    VOL=BPXLK1
           OA01001  OA01002  OA01003
       ECSA STORAGE: 1268496         OMVS  STORAGE:  4768248
```

Figure 39. Second example of D OMVS,ACTIVATE=SERVICE

Dynamically changing the BPXPRMxx parameter values

The SETOMVS command enables you to modify BPXPRMxx settings without reIPLing.

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 parameters 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 that is 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 reIPL. However, if you change the values, a reIPL will be necessary. For more information, see "Dynamically adding FILESYSTYPE statements in BPXPRMxx" on page 269.

SET OMVS=xx can be used to execute the ROOT, MOUNT, FILESYSTYPE, SUBFILESYSTYPE, and NETWORK statements in the BPXPRMxx member.

Dynamically changing certain BPXPRMxx parameter values

These parameters specify maximum values: MAXPROCSYS, MAXPTYS, IPCMSGNIDS, IPCSEMNIDS, IPCSHMNIDS, IPCSHMSPAGES, and SHRLIBRGNSIZE. 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 reIPL.

While you can specify the SHRLIBMAXPAGES parameter to specify a maximum value, it will be accepted but will not have any impact on the system. The value that you specify will never be reached, because user-shared library objects are no longer supported.

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 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 268, 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 reIPL.

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 reIPL.

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.

SHRLIBRGNSIZE and SHRLIBMAXPAGES

Use the D OMVS,L command to determine shared library usage and adjust your parameters.

If you specify the SHRLIBMAXPAGES parameter, it will be accepted but will not have any impact on the system. The value that you specify will never be reached, because user-shared library objects are no longer supported.

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 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.

Restriction: While the MAXPROCSYS, MAXPTYS, IPCMSGNIDS, IPCSEMNIDS, IPCSHMNIDS and IPCSHMSPAGES values can be changed dynamically, the changes are limited by the initial value that was used at IPL time, as described in "Dynamically changing certain BPXPRMxx parameter values" on page 267. SHLIBRGNSIZE and SHLIBMAXPAGES are not affected by this restriction.

SET OMVS=xx can be used to execute the ROOT, MOUNT, FILESYSTYPE, SUBFILESYSTYPE, and NETWORK statements in the BPXPRMxx member.

Dynamically adding FILESYSTYPE statements in BPXPRMxx

When you dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements, you can make the change permanent without having to reIPL by using the SETOMVS RESET command. 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 list shows examples of some of the more common configuration changes, adding the file system and adding sockets.

1. "Steps for activating the zFS file system for the first time" on page 269
2. "Steps for activating a single sockets file system for the first time" on page 270
3. "Steps for activating a multiple socket file system for the first time with Common INET" on page 271
4. "Steps for increasing the MAXSOCKETS value" on page 271
5. "Steps for adding another sockets file system to an existing CINET configuration " on page 272

Steps for activating the zFS file system for the first time

Perform the following steps to activate the zFS file system for the first time.

1. Set up a root file system by putting the following statement in BPXPRMxx.

   ```
   ROOT  FILESYSTEM ('OMVS.ROOT')
         TYPE ZFS
         MODE(RDWR)
   ```

   _____

2. Create a temporary BPXPRM*tt* member that has the following statement.

   ```
   FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
   ```

   _____

3. Dynamically add the statements to BPXPRM*tt*.

   ```
   SETOMVS RESET=(tt)
   ```

4. From TSO or the ISHELL, do the following:
   a. Unmount the current root file system.
   b. Mount the root data set as the new root file system.
   c. Mount any additional zFS data sets as needed.

5. Add the following statements to the BPXPRMxx member that is used on IPL:
   a. The FILESYSTYPE statement used in Step “2” on page 269.
   b. A ROOT statement for the root file system.
   c. MOUNT statements for the additional mounts that should be done initially.

When you are done, you have activated the zFS file system for the first time.

Activating a single sockets file system for the first time

About this task

This topic explains how to activate a single sockets file system for the first time. It uses the TCP/IP Socket File System for network sockets and also brings up support for local sockets.

Steps for activating a single sockets file system for the first time

Before you begin, you need to know what MAXSOCKETS value to use. The value used in the example might be different from the value that you want to use.

Perform the following steps to activate a single sockets file system for the first time.

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(64000)
   DOMAINNAME(AF_INET)  DOMAINNUMBER(2)

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

2. Dynamically add the statements to BPXPRM*tt*.

```
SETOMVS RESET=(tt)
```

3. Add the statements in Step “1” on page 270 to the BPXPRMxx member that is used on IPL.

When you are done, you have activated a single sockets file system for the first time.

Activating a multiple sockets file system for the first time with Common INET (CINET)

About this task

You will activate multiple sockets file systems for the first time with Common INET. In the example, two TCP/IP stacks are started.

Steps for activating a multiple socket file system for the first time with Common INET

Before you begin, you need to know what MAXSOCKETS value to use. The value used in the example might be different from the value that you want to use.

Perform the following steps to activate a multiple file system for the first time with Common INET (CINET).

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 nultiple TCP/IP stacks under Common INET */
SUBFILESYSTYPE TYPE(CINET)  NAME(TCPIP)  ENTRYOINT(EZBPFINI) DEFAULT
SUBFILESYSTYPE TIME(CINET)  NAME(TCPIP2) ENTRYPOINT(EZBPFINI)
```

_____

2. Dynamically add the statements to BPXPRM*tt*.

```
SETOMVS RESET=(tt)
```

_____

3. Restart TCP/IP.

```
S TCPIP
S TCPIP2
```

_____

4. Add the statements in Step “1” on page 271 to the BPXPRMxx member that is used on IPL.

_____

When you are done, you have activated a multiple sockets file system for the first time with Common INET.

Restriction: The names used in the example, TCPIP and TCPIP2, must match those used when configuring the associated products.

Specifying the maximum number of sockets

The MAXSOCKETS parameter in the BPXPRMxx member of SYS1.PARMLIB is used to specify the maximum number of sockets that can be obtained for a given file system type.

Steps for increasing the MAXSOCKETS value

The following steps assume that you want to make a permanent change to the MAXSOCKETS value without having to stop and then restart z/OS UNIX.

Restriction: This procedure can only be used if you have specified DOMAINNAME(AF_INET) or DOMAINNAME(AF_INET6). MAXSOCKETS is always set to 10000 for AF_UNIX and any MAXSOCKETS value specified on the NETWORK statement is ignored.

Before you begin, you need to know what MAXSOCKETS value you want to use, and you must have specified either DOMAINNAME(AF_INET) or DOMAINNAME(AF_INET6).

Perform the following steps to increase the MAXSOCKETS value.

1. Create a temporary parmlib member, BPXPRM*tt*, and add the following information:

```
NETWORK TYPE(INET) MAXSOCKETS(200000)
   DOMAINNAME(AF_INET)  DOMAINNUMBER(2)
```

   _____

2. Activate the updated BPXPRM*tt* member.

```
SETOMVS RESET=(tt)
```

   _____

3. To make the change permanent, update the MAXSOCKETS value in the BPXPRMxx member that is used on IPL.

   _____

When you are done, you have increased the MAXSOCKETS value.

You can add support for AF_INET6 to a running system for the first time. To do so, the NETWORK statement would specify DOMMAINNAME(AF_INET6) and DOMAINNUMBER(19). TCPIP would have to be recycled for this to take effect. You can add AF_INET6 in this way to an INET or as discussed in the next step to a CINET configuration. You can also change the MAXSOCKETS value for a CINET configuration with a similar procedure:

1. The TYPE() keyword of the NETWORK statement would specify the TYPE name of the CINET INET PFS, which was "CINET" in the previous examples.
2. INADDRANYPORT cannot be changed.
3. INADDRANYCOUNT can be increased for DOMAINNAME(AF_INET). INADDRANYCOUNT for AF_INET6 is ignored. The reserved port range for CINET is shared across both address families and the values are taken from the AF_INET statement. The maximum value allowed for INADDRANYCOUNT is 8000.
4. If you only want to increase MAXSOCKETS, and if INADDRANYCOUNT and INADDRANYPORT were specified on the NETWORK statement used during initialization, specify INADDRANYCOUNT and INADDRANYPORT the same way for the SET OMVS. If you omit them from the NETWORK statement, the default value of INADDRANYCOUNT=1000 is used.
5. Before INADDRANYCOUNT is increased, the PORTRANGE statement in the TCP/IP profile might need to be modified to reserve the additional ports for z/OS UNIX and the TCP/IP stacks recycled.

Adding another sockets file system to an existing Common INET (CINET) configuration

This topic explains how to start a second TCP/IP sockets file system.

Steps for adding another sockets file system to an existing CINET configuration

Before you begin, you need to know what socket file system you want to add.

Perform the following steps to add another sockets file system to an existing CINET configuration.

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

```
SUBFILESYSTYPE TYPE(CINET) NAME(TCPIP2) ENTRYPOINT(EZBPFINI)
```

_____

2. Dynamically add the statements to BPXPRM*tt*.

```
SETOMVS RESET=(tt)
```

_____

3. Start the TCPIP2 address space.

_____

4. Update the SUBFILESYSTYPE value in the BPXPRMxx member that is used on IPL.

_____

When you are done, you have added another sockets file systems to an existing Common INET configuration.

Tracing events

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.

Tracing events in z/OS UNIX

Use the CTnBPXxx parmlib members to specify events to be traced. Each member should specify one or more events. Keep the number of events small because tracing affects system performance. You can filter the events by address spaces, user IDs, and level of detail.

The CTRACE parameter of the BPXPRMxx member identifies the CTnBPXxx member to be used when the OMVS address space is initialized. Specify the size of the trace buffers in the CTnBPXxx member that is used when the system is IPLed. You can change the buffer size while z/OS UNIX is running. The minimum size of the buffer is 16 K and the maximum is 2047. The default is 128 M. If you need a different buffer size, change the buffer size (BUFSIZE) in a CTnBPXxx member and issue the following command:

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

You can dynamically change the buffer size with either of the following methods:

- Specify BUFFSIZE in a CTIBPXxx parmlib member. For example:

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

- You can also issue the following console command:

```
TRACE CT,xxxxM,COMP=SYSOMVS
R id,OPTIONS=(ALL),END
```

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

To trace DFSMS events, issue:

```
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
```

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 about interpreting a trace, see *z/OS DFSMSrmm Implementation and Customization Guide*.

Re-creating problems for IBM service

You might be asked to re-create a problem for IBM service to aid in diagnosis.

Tip: If you are re-creating a problem for IBM service, you should increase the OMVS CTRACE buffer size to 8 MB.

To increase the OMVS CTRACE buffer size to 8 MB, with the parmlib member specifying the required options, specify:

```
TRACE CT,8M,COMP=SYSOMVS,PARM=CTnBPXxx
```

As an alternative, you can change the parmlib member to specify the required buffer size. After you capture the dump for the problem, you can reset the trace buffer size to the original setting. The following example shows how to reset the trace buffer size to the original setting:

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

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

Filtering trace data

Use the JOBNAME= parameter on the TRACE CT command to filter trace data. The JOBNAME= parameter can be used for the OMVS CTRACE to trace data just for jobs that run with the specific user ID or user IDs that are specified in the JOBNAME list. It is important to note that this filtering is based on the user ID of the job, not its job name.

Displaying the status of the kernel or process

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 the one in "Steps for ending a specified process" on page 255.

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 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
```

Tip: Message BPXO040I contains a complete description of the DISPLAY OMVS output. To look up that message, go to BPX messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

You can cancel selected threads. For 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 system-wide limits specified in BPXPRMxx

You can display information about current system-wide 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
 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
 MAXUSERMOUNTSYS         15         20        100
 MAXUSERMOUNTUSER         7          8         10
 MAXPIPES                28         51      15360
```

An * displayed after a system limit indicates that the system limit was changed via a SETOMVS or SET OMVS= command. For the sysplex-wide limits, the command can be issued from any of the systems in the shared file system configuration environment, and the change can also be caused by subsequent OMVS initialization on the other systems.

The display output shows for each limit the current usage, high-water (peak) usage, and the system limit as specified in the BPXPRMxx member. The displayed system values might be the values as specified in the BPXPRMxx member, or they might 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. High water marks for process limits cannot be reset.

Taking a dump of the kernel and user processes

If you have a loop, hang, or wait state 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 might be associated with the problem.
- Any process address spaces that might be associated with the problem.
- Appropriate storage areas containing system control blocks (for example, SQA, CSA, RGN, TRT).

The steps are:

1. Use the DISPLAY operator command to display information about currently active address spaces and data spaces.
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, issue:

```
D A,OMVS
```

The output will be similar to the following:

```
d a,omvs
CNZ4106I 11.04.53 DISPLAY ACTIVITY 237
 JOBS     M/S    TS USERS    SYSAS    INITS   ACTIVE/MAX VTAM     OAS
00000    00007    00000      00029    00001    00000/00020       00006
 OMVS     OMVS     OMVS     NSW  *   A=000E   PER=NO   SMC=000
                                     PGN=N/A  DMN=N/A  AFF=NONE
                                     CT=000.572S  ET=02.05.49
                                     WKL=SYSTEM   SCL=SYSTEM   P=1
                                     RGP=N/A      SRVR=NO  QSC=NO
                                     ADDR SPACE ASTE=07F34380
                                     DSPNAME=SYSIGWB1 ASTE=03DD5480
                                     DSPNAME=SYSZBPXC ASTE=04EB1380
                                     DSPNAME=SYSZBPXU ASTE=04EB1300
                                     DSPNAME=HFSDSP04 ASTE=01837F80
                                     DSPNAME=HFSDSP03 ASTE=01837D80
                                     DSPNAME=HFSDSP02 ASTE=01837D00
                                     DSPNAME=HFSDSP01 ASTE=03DD5200
                                     DSPNAME=HFSDSPS1 ASTE=01837F00
                                     DSPNAME=SYSZBPX3 ASTE=01837E00
                                     DSPNAME=SYSZBPX2 ASTE=01EBE600
                                     DSPNAME=BPXSMBIT ASTE=01837B80
                                     DSPNAME=SYSZBPXX ASTE=07F39400
                                     DSPNAME=SYSZBPX1 ASTE=07F39380
                                     DSPNAME=BPXLATCH ASTE=01EBE400
```

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 that are associated with the kernel address space.

When the list of address spaces being dumped includes a dubbed address space or the kernel (OMVS) address space, the CTRACE buffers are automatically included in the dump and need not be explicitly added to a DUMP command or SLIP trap.

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, issue:

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

The output will be similar to the following example:

```
D OMVS,A=ALL

USER     JOBNAME  ASID        PID       PPID STATE
OMVSKERN BPXOINIT 002A          1          0 1WI
MVS      TCPIP    002B      65538          1 MR
TS65106  TS65106  0032          9          1 1RI
TS65106  TS65106  0032         10          9 1CI
  LATCHWAITPID=         0 CMD=-sh
```

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

A complete description of the D OMVS output can be found in message BPXO040I. To look up that message, go to BPX messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

Displaying global resource information

To display global resource serialization information to see possible latch contention, issue:

```
D GRS,C
```

This display might 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.

Displaying information about local and network sockets

You can display information about local and network sockets.

- To display information about network sockets (AF_INET and AF_INET6), use the TSO NETSTAT command. You can also use the z/OS UNIX `onetstat` or `netstat` command. (The `netstat` command is a synonym for the `onetset` command.) For more information about these commands, see *z/OS Communications Server: IP System Administrator's Commands*.
- To display information about AF_UNIX sockets (which are local sockets and therefore do not have any network connections), issue D OMVS,Sockets. For more information about D OMVS and an example of the display output, see SETOMVS command in *z/OS MVS System Commands*.

Detecting latch contention

This topic discusses latch contention.

For shared memory mutexes and conditional variables

To isolate contention problems for shared memory mutexes and condition variables, issue the D OMVS,SER operator command. It will display serialization data that you can use in problem determination.

- Each mutex and condition variable is identified by the shared memory ID and location of the shared memory object.
- If the object is in an above the bar shared memory segment, the location information indicates the address of the mutex or condition variable.
- If the object is in a below the bar segment, the location information indicates the offset within the shared memory segment. The offset is displayed, in this case, because each address space sharing a below-the-bar segment can map it at a different virtual address.
- For each mutex, the output shows the owner's TCB address, process ID and ASID, and the same for the waiters, if the information can be determined.
- For each condition variable, the output shows the same information for the owner and waiter of the mutex or conditional variable.
- If Language Environment is the caller of BPX1SMC, the user data represents the address of the Language Environment data area for the waiting or owning task.

If there is no contention, the output will be similar to the following:

```
BPXO057I 08.51.42 DISPLAY OMVS 284
OMVS     000E ACTIVE          OMVS=(6D)
                 UNIX SERIALIZATION REPORT
NO RESOURCE CONTENTION EXISTS
```

If there is contention, the display will be similar to the following:

```
BPXO0xxI 08.51.42 DISPLAY OMVS 284
OMVS     000E ACTIVE          OMVS=(6D)
                              UNIX SERIALIZATION REPORT
RESOURCE    #1:
  NAME=SHARED MUTEX   DATA: SHMID=00000648 OFFS/ADDR=0000000000002428

 JOBNAME ASID TCB        PID    USER DATA        EXC/SHR  OWN/WAIT
 DOMINO1 013A 008EF190 16777220 0000000024780148 EXC    OWN
 DOMINO2 02B2 008FA190 16908357 0000000024825220 EXC    WAIT
 DOMINO3 0206 008FF458 16973924 0000000024824778 EXC    WAIT
RESOURCE    #2:
  NAME=SHARED CONDVAR   DATA: SHMID=00000648 OFFS/ADDR=0000000000002458
JOBNAME ASID TCB      PID      USER DATA        EXC/SHR  OWN/WAIT
 DOMINO2 02B6 008FA190 16908357 0000000024825220 EXC    WAIT
 DOMINO3 0206 008FF458 16973924 0000000024824778 EXC    WAIT
 RESOURCE #0002 IS LOCKED BY:
  NAME=SHARED MUTEX   DATA: SHMID=00000648 OFFS/ADDR=0000000000002428
```

For user tasks

The kernel keeps track of latch contention caused by z/OS UNIX user tasks and takes action if a latch causing contention has been held for an excessive amount of time. If the problem is not corrected, an error message is issued, and you will be asked to issue the D GRS operator command to gather information about the latch resource, latch owners, and latch waiters. Try using the methods described in "Steps for ending a specified process" on page 255 to end the latch contention.

If the latch contention is not ended, then use the F BPXOINIT,RECOVER=LATCHES operator command. It ends user tasks that are holding latches for an excessive period of time. Only individual tasks are ended, not entire processes.

Note: F BPXOINIT,RECOVER=LATCHES might not be able to resolve latch hangs in the kernel address space.

Preallocating a sufficiently large dump data set

Because you are dumping multiple address spaces, multiple data spaces, and multiple storage data areas, you might 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.

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:

```
D D,O
```

The default value is 500 megabytes. To change this value, issue:

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

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

Taking dumps

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 about the DUMP command, particularly on specifying a large number of operands, see DUMP command in *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.

- If the kernel fails, both interactive processing in the shell and z/OS UNIX applications fail.
- If a file system type fails, 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.
- If a file system fails, programs might fail because some files cannot be used.

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

z/OS UNIX system 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 BPXF032D. In response, the operator or automation corrects the problem as indicated by previous messages and then enters R in reply to message BPXF032D.

If a file system type fails to initialize, the system normally issues message BPXF006I. If the failing file system type was specified with the option to prompt for restart (the default), the error that caused the problem can be corrected, and then the prompt responded to. If it was specified with the option to not prompt for restart, the system continues to run without that file system type, but requests to use the files in any file systems of that file system type will fail.

In rare cases, the initialization of a file system type might fail due to a programming or environmental error, such as a severe storage shortage in the kernel address space. The failure can occur before the file system type is initialized, and, on rare occasions, the BPXF006I message is not issued. In these cases, the severe programming or environmental error should be addressed first. After the severe condition that prevented the initialization of the file system type is resolved, the operator can manually initialize the file system type with the SETOMVS RESET=xx operator command.

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 cannot be opened

After a failure of a file system, the operator:

1. Restores the file system with the file system from the previous level.
2. Asks a superuser to logically mount the restored file system with a TSO/E MOUNT command.
3. Notifies all shell users that when they invoke the shell they will mount an earlier file system, telling them the mount point. (Use the `wall` command to broadcast a message to all shell users.)

Files added since the earlier file system was saved must be created again and then 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. A superuser who is defined with a home directory of `/`; (root) can also restore the file system. All work in progress when the failure occurred is lost and must be started from the beginning.

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 is ended or a user logs off. Therefore, it is possible that an IPC user might need assistance to do either of the following:

- 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:

```
ipcs -w
```

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

Another problem might 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 user in an IPC wait state out of the wait state.

To display which users are waiting for semaphores and message queues:

```
ipcs -w
```

Chapter 13. Managing processing for z/OS UNIX

Managing processing for z/OS UNIX involves tasks such as controlling printing and code page conversion.

List of subtasks

Subtask	Associated procedure
Making the Language Environment runtime library available through STEPLIB	“Steps for making the runtime library available through STEPLIB” on page 287

Controlling printing

Control printing by doing the following tasks:

- Designate printers to be used for shell users and applications.
- Set up default printers for each user.
- Control output print separators.

You can arrange for all printing to be done by one or two printers by assigning one or more output classes for all users. Then you and the users can look at the printer queues for those output classes to check for all output.

Infoprint Server provides an alternate version of the `lp` command, as well as related utilities. For more information, see *z/OS Infoprint Server User's Guide*.

Designating printers

Tell the application programmers the destinations or symbolic names for printers you specified in JES initialization statements. The *dest* option of the `lp` command uses the same destinations as the DEST parameter in the OUTPUT JCL statement.

The *dest* option on `lp` can be:

- LOCAL for any installation printer.
- A destination that is defined in a JES2 DESTID initialization statement.
- Omitted. The system uses the default printer.

Setting up default printers

Each user has a number of default printers specified in different ways, as shown in Table 34 on page 283. The system will use printer number 1, if designated; if not, the system will use printer number 2; and so on.

Table 34. Default printers

Printer number	Printer designation	Specified by
1	The printer in the *dest* option of the `lp` shell command, or the printf() or fprintf() functions.	User or application programmer
2	The printer LPDEST environment variable	User or system programmer
3	The printer PRINTER environment variable	User or system programmer

Table 34. Default printers (continued)		
Printer number	**Printer designation**	**Specified by**
4	The printer in the RACF user profile. It is specified by the DEST parameter of the RACF ADDUSER or ALTUSER command.	Security administrator

Controlling output print separators

JES controls the print separators, also called *cover pages* and *banner pages*, for SYSOUT output for all users, including z/OS UNIX users.

To place a user's name and address in the print separator for forked processes, specify the user's name and address in the WORKATTR segment of the RACF user profile. See "Defining z/OS UNIX users to RACF" on page 51.

Controlling code page conversion

For an overview of character sets and code pages, refer to *National Language Support Reference Manual, Volume 2,* SE09-8002.

A code page for a character set determines the graphic character that is produced for each hexadecimal code. The code page is determined by the programs and national languages that are being used.

The z/OS UNIX Application Services can process data in the following code pages:

- Any of the EBCDIC Latin 1 Country-Extended Code Pages
- Japanese (Latin) Extended Code Page 01027, which defines single-byte encodings for character set 01172 (Japanese Extended EBCDIC/PC Common)
- Japanese Combined Code Page 00939, which is the combination of code page 01027 and code page 00300. Code page 00300 (Japan [Kanji]–Host, DBCS) defines DBCS encodings for character set 00370 (IBM Japanese Graphic Character Set, Kanji)

Data intended for processing by the z/OS shell might require conversion to one of the preceding code pages. This data might be encoded in:

- Latin 1 code page 00500, which is used for Systems Application Architecture® (SAA).
- An ASCII code page, for example, for a file from a workstation. A source program on a tape archive (TAR) tape might be stored in the ASCII code set.
- Code page 00293, which the z/OS XL C/C++ compiler can optionally use.
- Code page 00290, Japanese (Katakana) single byte.
- Code page 00930, the Japanese combined code page (code page 0290 plus DBCS code page 300).

For code page 00037, only two characters are different from code page 01047:

- Right bracket (])
- Left bracket ([)

If you have characters from the preceding list in your data, you must convert from one code page to another when, for example, you are doing one of the following tasks:

- Transferring files between a workstation and the file system.
- Copying data between an MVS data set and a UNIX file.
- Placing data in SYSOUT data sets.
- Passing JCL path name data to programs, unless the name contains only characters in the portable file name character set.
- Passing JCL parameters and path names to a shell invoked from a batch program, unless the parameters and names contain only characters in the portable file name character set.

- Using the `lp` command to print. You need to convert the data before you send it to the printer.
- Using the `pax` command.

Converting single-byte data

If MVS data is single-byte, you can specify the conversion at the same time that you copy the data.

You must specify the CONVERT operand in the TSO/E OCOPY, OGET, OPUT, OGETX, and OPUTX commands to convert the data that the command is copying. Copying can be from data sets or UNIX files and to data sets or UNIX files.

OGET in *z/OS UNIX System Services User's Guide* provides examples of using OGET to convert data with a PDSE member or a sequential data set.

Converting double-byte data

Double-byte data that is already in a supported DBCS code page, such as IBM-939, does not need to be converted.

DBCS data not in code page IBM-939 must be converted to IBM-939 with the `iconv` command so that it can be processed in the z/OS UNIX environment.

If the data is in a code page not supported by the shell, you can copy the data with the OCOPY command first and then convert it using the `iconv` command. Or you can convert the data with the TSO/E ICONV CLIST and then copy it using the OCOPY command.

Using character conversion tables

A character conversion table is a table that converts one or more characters to alternative characters using hexadecimal encoding for the character sets. The character sets are defined in code pages. IBM supplies these character conversion tables as members in SYS1.LINKLIB:

- BPXFX100 (code page 00037 to and from 01047 for non-APL 3270)
- BPXFX111 (null conversion)
- BPXFX211 (code page 00037 to and from 01047 for APL 3270)
- BPXFX311 (ASCII code page 008859 to and from 01047)

In particular, for the OMVS command, BPXFX100 is the default conversion table. As shipped, FSUMQ000 is an alias for BPXFX100. To change the OMVS default table, move the FSUMQ000 ALIAS to the new default, or rename the new default table to FSUMQ000.

Users who need different conversion tables can manually override the default table by using the CONVERT operand. In addition, the system programmer can write a REXX exec or CLIST to invoke the OMVS command with the proper table.

The source for these members is shipped in SYS1.SAMPLIB. If you need to change them, see “Customizing code page conversion” on page 286.

Example of data conversion specified by a user

Use the OPUT command for conversion of single-byte data. You can copy a sequential data set or partitioned data set member to a UNIX file. Code page conversion is an option.

For example. issue:

```
OPUT WORKLOAD.TOTALS(OCT17) 'u/turbo/wkld/totals/oct17' TEXT CONVERT(YES)
```

Result: The user ID TURBO copied a member from a PDSE into a file. The partitioned data set member OCT17 was copied from the data set TURBO.WORKLOAD.TOTALS to a text file with the path name `/u/turbo/wkld/totals/oct17`. Data was converted from the z/OS UNIX country-extended code page to code page 01047, using the default conversion table because YES was specified. To use a different

conversion table, specify its name—for example, BPXFX311 for conversion from the ASCII conversion table. If you do not want conversion, omit the CONVERT operand.

Customizing code page conversion

If the installation has special conversion needs for single-byte data conversion, create the needed character conversion table by customizing an existing table:

1. Copy the assembler source for a table from SYS1.SAMPLIB into a new data set.
2. Edit the new data set to customize the table. SYS1.SAMPLIB(BPXFX100) shows the format of the conversion tables and, in its prolog, gives instructions and examples of how to edit a table.
3. Assemble the table.
4. Link-edit the table into a load module in SYS1.LINKLIB or another partitioned data set.

Each table is 1792 bytes long and contains the 8-bit codes that the system substitutes for characters in the input data set or file. Each table contains nine sections; you might have to change the data in all nine sections. Each member has a TO and a FROM subtable:

- The TO subtable is used to translate data from another code page to 01047.
- The FROM subtable is used to translate data from code page 01047 to another code page.

Example of code page conversion of OMVS command

For customization, suppose you change a table, name it WCOFXCHG, and place it in a SYS1.LINKLIB member. Then you would use the following OMVS command to invoke the shell, with SYS1.LINKLIB understood as the location of WCOFXCHG:

```
OMVS CONVERT((WCOFXCHG))
```

To avoid conflicts in the names of modules containing tables, begin your name with letter *K* through *Z*; letters *A* through *J* are reserved for IBM use.

Managing z/OS UNIX in relation to other processing

The IMS batch message processing program (BMP) can request z/OS UNIX services. However, the following applications cannot request z/OS UNIX services:

- Customer Information Control System/ESA (CICS/ESA): A CICS transaction cannot access a file in a file system, because the access would put CICS in a wait. Also, other z/OS UNIX functions could not be used and would abnormally end CICS.
- Information Management System/ESA (IMS/ESA®): An IMS application cannot access a file in a file system, because the access would put the IMS application in a wait. Also, other z/OS UNIX functions could not be used and would abnormally end the IMS application.

JES2 processing

In a JES2 multi-access spool (MAS) complex, a z/OS UNIX application might experience the following situation:

The system can convert the job on one system and interpret it on another.
If all systems do not have z/OS UNIX, the job processing can begin on a system with z/OS UNIX installed and started, but continue on a system without z/OS UNIX installed.

If a job requires z/OS UNIX or a file system, use a JES2 /*JOBPARM statement with a SYSAFF keyword to direct the job to the correct system.

If you need to bring down JES2, there might 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
```

For more information, see “Partial shutdowns for JES2 maintenance” on page 260.

JES3 processing

In a JES3 global or local configuration, a z/OS UNIX application might experience the following situation: one system can complete conversion and interpretation and another system can run the job. If all systems do not have z/OS UNIX, the job might be assigned to a system without z/OS UNIX. In this case, the job will fail. To prevent this problem, use a JES3 `//*MAIN` statement with a SYSTEM keyword to direct the job to a system with z/OS UNIX.

Accessing the Language Environment runtime library

For most z/OS UNIX applications, the Language Environment runtime library (SCEERUN and SCEERUN2) is needed. The SCEERUN and SCEERUN2 data sets must be placed in the LNKLST or the LPA list, but can also be accessed via STEPLIB. (A STEPLIB is a set of private libraries that are used to store a new or test version of an application program, such as a new version of a runtime library.)

If the Language Environment runtime library is heavily used at your installation, consider placing the SCEELPA data set (which contains key modules) in the LPA list for better performance. See “Improving performance of runtime routines” on page 357.

To test new levels of the runtime libraries, you can make the Language Environment runtime library available through STEPLIB as described in “Steps for making the runtime library available through STEPLIB” on page 287.

Steps for making the runtime library available through STEPLIB

Before you begin, the Language Environment runtime library must reside in the LNKLIST or the LPA list. However, if you want to test a new version of the runtime library, then make it available through STEPLIB.

Do not include a STEPLIB DD statement in the BPXAS procedure. Doing so can lead to recursive 0C4 abends when the BPXAS procedure is processing a fork or create request.

Perform the following steps to make the runtime library available through STEPLIB.

1. Add the SCEERUN and SCEERUN2 data sets on a STEPLIB DD statement to the OMVS startup procedure found in PROCLIB.

   The STEPLIB data set is propagated to BPXOINIT and the `/usr/sbin/init` program including all programs that it invokes by using fork or exec.

   _____

2. Add the SCEERUN and SCEERUN2 data sets to your TSO/E logon procedure by concatenating them to the ISPLLIB DD statement (if it exists) and then concatenating them to the STEPLIB DD statement (if it exists). You can also use the TSOLIB function to add the SCEERUN and SCEERUN2 data sets.

   After you have added the SCEERUN and SCEERUN2 data sets, the TSO/E OMVS command can use them.

   _____

3. Add the following statement to the `/etc/rc` file:

   ```
   export STEPLIB=hlq.SCEERUN:hlq.SCEERUN2
   ```

   Daemons started in `/etc/rc` use the SCEERUN and SCEERUN2 data sets.

   _____

4. In /etc/profile, remove:

   ```
   if [ -z "$STEPLIB" ] && tty -s;
   then
       export STEPLIB=none
       exec sh -L
   fi
   ```

   and replace with

   ```
   export STEPLIB=hlq.SCEERUN:hlq.SCEERUN2
   ```

   This is used when issuing commands and utilities in the shell environment.

   If a few interactive users need a special version of the runtime library, the STEPLIB environment variable can be set in the $HOME/.profile for each of these users.

   _____

5. Add the SCEERUN and SCEERUN2 data sets on a STEPLIB DD statement to any job invoking BPXBATCH.

   _____

6. Add the SCEERUN and SCEERUN2 data sets to the STEPLIBLIST statement of the BPXPRMxx parmlib member. The SCEERUN and SCEERUN2 data sets must be APF-authorized.

   _____

When you are done, you have made the Language Environment runtime library available through STEPLIB.

For BPXBATCH processing, you still must specify the SCEERUN and SCEERUN2 data sets on a STEPLIB DD statement even though the RUNOPTS parameter has been set in the BPXPRMxx member.

Fastpath support for System Authorization Facility (SAF)

System Authorization Facility (SAF) provides a system interface that conditionally directs control to RACF or any other security product when a request is received from a resource manager. To improve the performance of security checking done for z/OS UNIX, define the BPX.SAFFASTPATH FACILITY class profile. Defining the profile reduces overhead when doing z/OS UNIX security checks for a wide variety of operations. These checks include file access checking, IPC access checking, and process ownership checking.

Restriction: BPX.SAFFASTPATH FACILITY only applies to HFS. As of V2R5, HFS is no longer supported.

When the BPX.SAFFASTPATH FACILITY class profile is defined, the security product is not called if z/OS UNIX can quickly determine that file access will be successful. When the security product is bypassed, better performance is achieved, but the audit trail of successful accesses is eliminated.

If the security product is called, it is still possible that access will be successful, and that audit records will be created; for example, when the permission bits do not grant access, but UNIXPRIV authority, or an access control list, does.

Be aware that auditing successful accesses can generate enormous amounts of audit records, particularly for directory searches.

Enabling the SAF fastpath support

If the BPX.SAFFASTPATH FACILITY class profile is defined when the system is IPLed, the SAF fastpath support is enabled. If it is defined after the system is IPLed, you must issue the SETOMVS or SET OMVS operator command to activate the fastpath support. You can also start the refresh by issuing the following command, where *xx* represents an empty BPXPRMxx member.

```
SET OMVS=xx
```

Users do not need to be permitted to the BPX.SAFFASTPATH profile.

To define the BPX.SAFFASTPATH profile, issue the following RACF command:

```
RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE)
```

If your installation uses the IRRSXT00 exit to control access to the file system, do not define the BPX.SAFFASTPATH profile.

Disabling the SAF fastpath support

To disable the fastpath support, remove the BPX.SAFFASTPATH FACILITY class profile and then issue the SET OMVS or SETOMVS operator command. You do not need to reIPL.

Determining problem causes

If a problem occurs, the system might write a dump and issue messages or an abend. Collect the problem data and determine the cause of the problem as you would for any system problem. For information about how to take a dump, see "Taking a dump of the kernel and user processes" on page 276.

Abends

z/OS UNIX services issues system completion codes: abend codes EC6 and 422.

All 422 abends and some EC6 abends might not be accompanied by an SVC dump, because the IBM-supplied IEASLP00 parmlib member contains SLIP commands to suppress the dumps.

Some abends are a normal result of a `kill` shell command, an `exec` shell command or program function, or the ending of a process. Others are caused by errors.

Return codes and reason codes

If a z/OS system service fails, a failing return code and reason code is sent. Reason codes are unique and should supply enough information to debug the problem. You can set a slip trap on a specific reason code to gather further diagnostic data. For information about setting the slip trap, see the section on reason codes listed by value in Reason codes in *z/OS UNIX System Services Messages and Codes*.

Messages

z/OS UNIX issues messages with the following prefixes:

BPX
Messages from the System Services component

FDBX
Messages from the dbx debugger

FOM
Messages from the Application Services component

FSUM
Messages from the Shell and Utilities

DFSMS issues messages with the prefix IGD.

Messages to the operator and system programmer have identifiers; messages from the shell to the interactive user do not have identifiers.

You can refer to the following publications:

- For the BPX messages, see BPX messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.
- The IGD messages are in IGD messages in *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.
- For the FDBX, FOM, and FSUM messages, see Shell and Utilities messages in *z/OS UNIX System Services Messages and Codes*.

Writing messages to a job log file

You can use the _BPXK_JOBLOG environment variable to specify that messages be written to a job log. Set it to one of these values:

nn
Job log messages are written to open file descriptor *nn*.

NONE
Job log messages are not written. NONE is the default.

STDERR
Messages are written to the standard error file descriptor, 2.

You can change the file that is used to capture messages by calling the oe_env_np (BPX1ENV) service and specifying _BPXK_JOBLOG with a different file descriptor. Message capturing is turned off if the specified file descriptor is marked for close on a fork or exec. Message capturing is process-related. All threads under a given process share the same job log file, and any thread under that process can initiate message capturing.

Multiple processes in a single address space can each have different files active as the JOBLOG file. Some or all of them can share the same file, and some processes can have message capturing active while others do not.

When the file that is used as a job log is shared by several processes (for example, by a parent and child), the file should be opened for append. If the file is not opened, unpredictable results might occur. Only files that can be represented by file descriptors can be used as job log files; MVS data sets are not supported.

Message capturing is propagated on a fork or spawn.

- If a file descriptor was specified, the physical file must be the same before message capturing can continue in the forked or spawned process.
- If STDERR was specified, the file descriptor can be remapped to a different physical file. You can override message capturing on exec or spawn by specifying the _BPXK_JOBLOG environment variable as a parameter to the exec or spawn. Message capturing only works in forked (BPXAS) address spaces.

Messages that would normally go to the JESYSMSG data set are captured, but messages that go to JESMSGLG are not captured.

Component identifiers

Table 35 on page 290 lists the component identifiers that are used in dumps and symptom strings.

Table 35. List of component identifiers that are used in dumps and symptom strings

Component identifier	Code	Module prefix
SCPX1 SCPX4 SCPX6	z/OS UNIX System Services	BPX FOM BOP
SCPX2	Shell and Utilities, shell initialization, TSO/E OMVS command, and the **c89** shell command	FSUM
SCPX3	dbx debugger	FDBX
5696EFS00	zFS physical file system	IOE

Formatting dumps

To format problem data in a stand-alone dump or SVC dump, use the interactive problem control system (IPCS) OMVSDATA subcommand. OMVSDATA is not useful in an SYSMDUMP dump or a core dump, which

the system writes for an application program, because these dumps do not contain the z/OS UNIX programs or data structures.

See the following:

- *z/OS MVS System Codes* for the abend codes
- *z/OS MVS IPCS Commands* for the syntax of the IPCS OMVSDATA command
- *z/OS MVS Diagnosis: Reference* for information about formatting an SVC dump, and for general problem determination procedures.
- *z/OS MVS Diagnosis: Tools and Service Aids* for SYSMDUMP and SYSABEND dumps that are produced by applications

Diagnosing problems

If the problem is in the shell or debugger, the system treats it as an application problem.

See TERMTHDACT in *z/OS Language Environment Programming Reference*. There are suboptions for the TERMTHDACT runtime option that enables you to specify the amount of dump data that is to be collected.

If you specify a dump by setting the _BPXK_MDUMP environment variable, you do not have to allocate a SYSMDUMP data set for the TSO/E session. The dump is written to either the MVS data set or the specified z/OS UNIX file.

If the _BPXK_MDUMP environment variable is not set, then you can specify a dump by allocating a SYSMDUMP data set for the TSO/E session. The system then does the following:

- Creates a file in the user's working directory.
- Names it `coredump.`*pid*, where *pid* is the process ID for the process being dumped. It is in hexadecimal format.
- Writes a core dump in the file. The core dump is a SYSMDUMP dump.

Setting _BPX_SHARAS to NO causes the output to go to a UNIX file. If it is set to REUSE (or YES), the dump is written directly to the MVS file, and the `coredump.pid` file is not created.

To use the core dump, follow these steps:

1. Copy the file into an MVS data set with a record length (LRECL) of 4160, by entering a TSO/E OGET command.
2. Use IPCS to analyze the dump. (Use the IPCS subcommands STATUS and SUMMARY FORMAT CURRENT.)

You can dynamically request a SYSMDUMP by using the SIGDUMP signal. Use the _BPXK_MDUMP environment variable to specify where the SYSMDUMP is to be written to. You can also use F BPXOINIT,DUMP=*pid* to request a SYSMDUMP. A SIGDUMP signal is then sent to the specified process. For both the SIGDUMP signal and the F BPXOINIT,DUMP command, the _BPXK_MDUMP environment variable must be set to an MVS data set name. If it is set to a UNIX file name or defaulted to OFF, then both the SIGDUMP signal and the F BPXOINIT,DUMP command might be ignored.

Diagnosing problems in application programs

The **dbx** debugger helps in debugging application programs that are written in the C language. With the debugger, the application programmer can set breakpoints at source statements and function entry points, display and modify storage using program variable names rather than absolute storage addresses, trace execution at the source statement level, and so on.

Diagnosing hangs during z/OS UNIX initialization

If there is a hang during initialization, the hang is likely to occur during the initialization process (`/etc/init` and `etc/rc`).

If you receive message BPXP006E indicating that z/OS UNIX is being initialized, you can check the `/etc/log` file to see what the last command processed from `/etc/rc` was. This might help you determine the cause of delay or hang.

The sample `/etc/rc` file that is shipped with z/OS UNIX includes the `set -v -x` command. That command specifies that shell input lines are to be printed to `/etc/log` as they are run, in addition to commands and their arguments.

Tip: If you are a superuser (permission to BPX.SUPERUSER is not sufficient), you can view `/etc/log` during a hang in `/etc/rc` by starting a shell from a superuser and issuing the following command:

```
cat /etc/log
```

Chapter 14. Managing the temporary file system (TFS)

The temporary file system (TFS) is an in-memory physical file system that supports in-storage mountable file systems and is not written to DASD. Putting the temporary data in a separate file system makes it easier to manage the space used by temporary files. Typically, a temporary file system runs in the kernel address space, but it can be run in a logical file system (LFS) colony address space.

Tip: Run TFS in a colony address space if more space is needed than can fit in an address space. When it is run in a colony address space, you can use the STOP and MODIFY commands. For more information, see "Running a physical file system in a colony address space" on page 38.

If TFS is executing in either a colony PFS address space or in the OMVS kernel, you can use the F OMVS,PFS=<*tfs*> system command to view or alter the TFS global settings. <*tfs*> is the value that is specified in the TYPE() parameter of the FILESYTYPE statement in the BPXPRMxx parmlib member.

If the TFS is executing in a colony PFS address space, you can use the F TFS command where TFS is the value that is specified in the ASNAME() parameter of the FILESYSTYPE statement.

Tip: Use F OMVS,PFS=TFS when managing the temporary file system.

Features of the TFS

The temporary file system includes these features:

- Attributes unique to z/OS UNIX are supported (for example, external symbolic links and the create time attribute) and is not part of a branded product.
- Access control lists (ACLs) are supported. The number of ACL entries that TFS supports is limited by the block size. Each ACL takes one TFS block. For example, if the TFS block size is set to 4K (the default is -b0), the number of ACL entries in any ACL is limited to approximately 500. For more information about ACLs, see "Using access control lists (ACLs)" on page 89.
- You can set up basic partitioned access method (BPAM) access to files in the TFS. Each TFS directory is treated as if it were a PDSE or PDS directory.

Security considerations

If you require a high level of security in your z/OS system and do not want superusers to have access to z/OS resources such as SYS1.PROCLIB, read these sections:

- "Comparing UNIX security and z/OS UNIX security" on page 311.
- "Establishing the correct level of security for daemons" on page 313.

Creating the TFS

The TFS is automatically mounted if the kernel is started in minimum mode. In this environment, the TFS is the in-storage file system and it defaults to the root file system. If it is to be used in other situations, it is made available by mounting. Because the TFS is a temporary file system, all data that is stored in the file system is discarded after it is unmounted. If you mount another TFS, that file system has only dot (`.`) and dot-dot (`..`) and nothing else.

If you are using kernel services in full function mode, you might want to mount a temporary file system over `/tmp`. If you do, it can be used as a high-speed file system for temporary files. However, you cannot

recover **vi** files if the system goes down because **vi** writes temporary files to TMPDIR (/tmp by default). To recover these files, use the **exrecover** command, which automatically runs from /etc/rc.

Restriction: You cannot mount a TFS using a DDname. If the TFS is unmounted, all data that is stored in a TFS is lost; when remounted, the file system has only dot(.) and dot-dot(..) entries.

Portable Software Instances installation jobs define the following FILESYSTYPE definition in SYS1.PARMLIB(BPXPRMFS):

```
FILESYSTYPE TYPE(TFS) ENTRYPOINT(BPXTFS)
```

Edit SYS1.PARMLIB(BPXPRMFS), that is, add the mount statement to mount a file system at the /tmp mount point. For example, you can add the following mount statement under the FILESYSTYPE TYPE(TFS) definition:

```
MOUNT FILESYSTEM('/TMP')
MOUNTPOINT('/tmp')
TYPE(TFS)
PARM('-s 10')
```

(-s 10) allocates 10 MB of storage

The following example shows the **mount** command for a TFS. Typically this specification would be in BPXPRMxx.

```
FILESYSTYPE TYPE(TFS) ENTRYPOINT(BPXTFS)
MOUNT FILESYSTEM('/TMP') TYPE(TFS)
MOUNTPOINT('/tmp')
PARM(' -s 10')
```

Note:

1. FILESYSTEM must be a unique name for the file system. Using the path name of the mount point makes it easier to understand output that is produced by commands such as **df**.
2. PARM specifies how much virtual storage the TFS can use. It can also be used to specify other information as listed in “Parameter key options for the mount statement and mount commands” on page 295. If PARM is omitted or is not valid, the TFS defaults to 1 MB. If the mount request specifies a size in megabytes that is too large for the address space, the request fails with an EMVSERR (9D) error. Try the request again, using a smaller value.
3. MODE is either RDWR or READ.
4. To specify that the temporary files are to be written to a specified directory instead of TMPDIR, use the TMP_VI environment variable.

Checking the size of the TFS

Because the TFS uses virtual storage, make sure that your real and auxiliary storage configurations are large enough to accommodate the total size of the file systems to be mounted on the TFS. The maximum file size that TFS supports is a function of the TFS block size. To find the maximum file size, refer to Table 36 on page 294, which shows the approximate maximum file sizes that are based on the block size factor.

Table 36. List of maximum file sizes that the TFS supports

Block size factor size	File size
0	2 GB
1	25 GB
2	240 GB
3	2 TB
4	17 TB

Parameter key options for the mount statement and mount commands

The parameter key options for the mount statement and mount commands are as follows:

-3
Specifies that the file system is to be allocated in 31-bit mode regardless of system capabilities.

-b <*block*>
<*block*> specifies the blocking factor that is used to set the size of a TFS block.

The following chart shows how <*block*> relates to the TFS block size:

0
4 K

1
8 K

2
16 K

3
32 K

4
64 K

Default: 0

–c <*cache*>
<*cache*> is the amount of buffer storage in MB that TFS uses in the 32-bit address range to support a 64-bit TFS. This parameter is ignored for file systems that are allocated in the 31-bit range. The default is calculated based on the TFS block size such that the numbers of cache buffers is 256.

-ea *count*
Allows the TFS file system to automatically grow *count* times. The TFS grows 1-K blocks each time it grows. For the default 4-K block size, the TFS grows 4 MB. The extension occurs when the file system fills up, before the file system full message.

Default: 0

Restriction: The sum of auto-extend and manual extend cannot exceed 500.

-em *count*
Allows the TFS file system to manually grow *count* times. The TFS grows 1-K blocks each time it grows. For the default 4-K block size, the TFS grows 4 MB. If `-em` is specified, any excess block allocation space is added to *count*.

Default: The calculated number that is based on the excess block allocation space for the file system.

Restriction: The sum of auto-extend and manual extend cannot exceed 500.

-fsfull(*threshold*,*increment*)
Sets FSFULL monitoring to the specified values. When specified, this option overrides the default setting.

- *threshold* is a number in the range 1-99. When the temporary file system is *threshold* percent full, a message is issued.
- *increment* is a number in the range 1-99. When the temporary file system becomes *increment* full or empty, any threshold messages that were previously issued is deleted.

`-fsfull` can be specified in uppercase or lowercase.

Default: fsfull (99,5)

-g <*group*>
<*group*> is the numeric GID that is to be assigned to the root directory of the file system. The default is 0.

-nofsfull
Sets FSFULL monitoring to the TFS default of `fsfull(99,5)`. When specified, this option overrides the TFS default setting.

-p <*perm*>
<*perm*> is the permission bits in octal to be assigned to the root directory of the file system. The default is 0777.

-s <size>
<*size*> is the number of MB for the file system. If the value specified is larger than the size that can be supported, the maximum size is used. The default is 1.

Note: If the size specified for the file system size is too large, the maximum file system size is used and a message is not issued. If the extend factors will result in the file system size becoming larger than the maximum size, the extend factors are ignored. A message is not issued and the file system will not have extend capability.

-u <uid>
<*uid*> is the numeric UID that is to be assigned to the root directory of the file system. The default is 0.

Invalid mount options are ignored.

Parameter key options for the FILESYSTYPE statement

The parameter key options for the FILESYSTYPE statement are as follows:

-ea *count*
Allows the TFS file system to automatically grow *count* times. The TFS grows 1 K blocks each time it grows. For the default 4 K block size, the TFS grows 4 MB. The extension occurs when the file system fills up, before the file system full message. When specified, this option overrides the TFS default setting.

Default: 0

Note: The sum of auto-extend and manual extend cannot exceed 500.

-em *count*
Allows the TFS file system to manually grow *count* times. The TFS grows 1 K blocks each time it grows. For the default 4 K block size, the TFS grows 4 MB. If `-em` is specified, any excess block allocation space is added to *count*. When specified, this option overrides the TFS default setting.

Default: 0

Note: The sum of auto-extend and manual extend cannot exceed 500.

-fsfull(*threshold*,*increment*)
Sets FSFULL monitoring to the specified values. When specified, this option overrides the default setting.

- *threshold* is a number in the range 1-99. When the temporary file system is *threshold* percent full, a message is issued.
- *increment* is a number in the range 1-99. When the temporary file system becomes *increment* full or empty, any threshold messages that were previously issued is deleted.

`-fsfull` can be specified in uppercase or lowercase.

"FILESYSTYPE" on page 21 contains more information about FILESYSTYPE. Complete reference information for FILESYSTYPE is in BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.

Monitoring space in the TFS

The storage administrator or system programmer can monitor the space in a file system by mounting a file system with the FSFULL mount parameter. Status messages are issued as the system fills up. For

example, by specifying `mount parm('FSFULL(70,10)')`, message BPXTF009E is displayed when the file system is 70 percent full. Another message is issued when the file system is 80 and 90 percent full.

You can use the F OMVS,PFS=TFS,FSFULL command to change the default FSFULL value. This value applies to the subsequent TFS mounts that do not specify FSFULL.

Determining the default setting for FSFULL monitoring

To display the current default setting for the `-fsfull`, `-ea`, and `-em` options, issue:

```
F OMVS,PFS=TFS,Q
```

In addition to displaying the default settings, information is also displayed about each mounted TFS.

Changing the default FSFULL setting

To change the default FSFULL setting:

```
F OMVS,PFS=TFS,FSFULL(threshold,increment)
```

threshold
Specifies a number in the range 1-99. TFS issues a message when a TFS file system is *threshold* percent full.

increment
Specifies a number in the range 1-99. TFS updates or deletes a threshold message that was issued when the file system fills or empties *increment* percent.

Dynamically extending the size of the TFS

To manually extend the size of the TFS, the maximum size or a maximum number of extensions can be specified. For example:

```
F OMVS,PFS=TFS,GROW filesysname
```

where *filesysname* is the name of the file system to be extended. Each extension is 1 K blocks.

The default extend settings can be changed. For example:

```
F OMVS,PFS=TFS,EA number
F OMVS,PFS=TFS,EM number
```

where *number* is the number of automatic or manual extends allowed for a file system that is subsequently mounted without using the `-ea` or `-em` parameters.

Restriction: The sum of auto-extend and manual extend cannot exceed 500.

Stopping the temporary file system (TFS)

The recommended mechanism to stop a TFS that is executing in either the OMVS address space or in a colony address space is to use the F OMVS,STOPPFS=TFS command, where TFS is the value that is specified in the TYPE parameter of the FILESYSTYPE statement.

For a colony TFS, you can use the MODIFY TFS command to stop a TFS running in a colony address space. The MODIFY TFS command can be used to force a TFS to stop or terminate even if TFS file systems are mounted.

Restriction: The MODIFY TFS command is not supported if TFS runs in the OMVS kernel address space.

The complete syntax for the MODIFY TFS command is as follows:

```
F TFS,{STOP}
      {TERM}
      {FORCESTOP}
```

```
      {FORCETERM}
```

TFS
The name of the TFS to be stopped.

STOP
This is the same function as the STOP command. If no TFS file systems are mounted, this command causes TFS to exit. A write to operator with reply (WTOR) is issued allowing TFS to be restarted.

TERM
If no TFS file systems are mounted, this command causes TFS to exit without prompting to restart the TFS. You can issue the SETOMVS RESET=(*xx*) command to start another TFS.

FORCESTOP
Similar to STOP, issuing this command causes TFS to terminate even if there are mounted TFS file systems.

FORCETERM
Similar to TERM, issuing this command causes TFS to terminate even if there are mounted TFS file systems.

Restriction: The F OMVS,PFS command does not support the STOP, TERM, FORECESTOP, and FORCETERM functions.

Using the TFS in a shared file system

A shared file system can include a temporary file system.If you are using a TFS for `/tmp`, the FILESYSTEM name must be different because each system requires its own copy. For example:

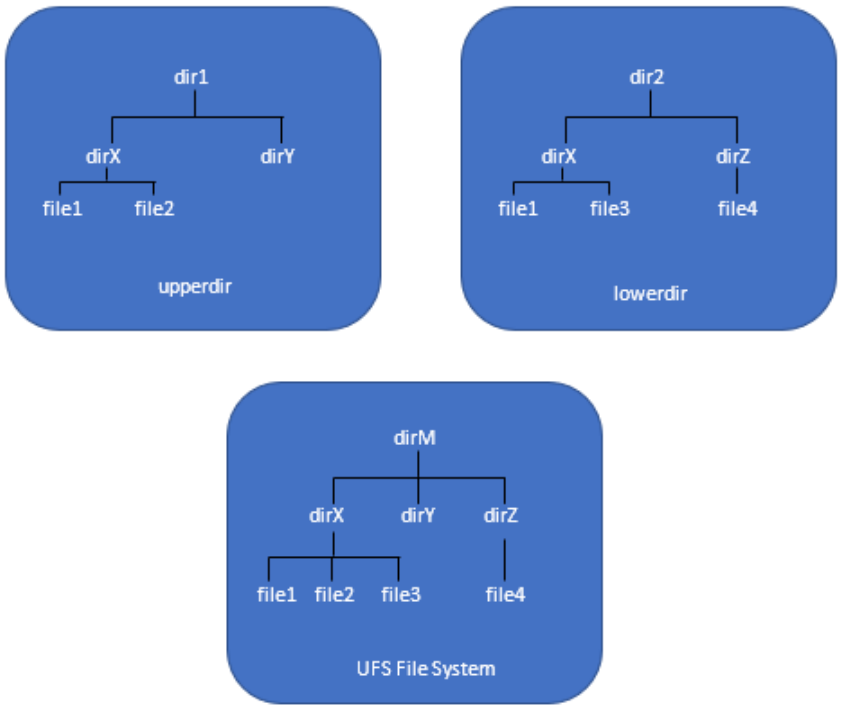
```
MOUNT FILESYSTEM('/TMP&SYSNAME.')
TYPE(TFS) MODE(RDWR) UNMOUNT
MOUNTPOINT('/&SYSNAME/tmp')
PARM('-s 10')
```

Because &SYSNAME is different on each system, `'/TMP&SYSNAME.'` has a different file system name on each system.

Chapter 15. Managing the union file system (UFS)

The union file system (UFS) provides a merged view of two or more directories. This merged view is obtained by accessing the mount point of the union file system. As an example, given the two directory hierarchies in the `upperdir` and `lowerdir`, the mount point hierarchy (`dirM`) is the root of the UFS file system that would result from the following **`mount`** command.

```
mount -t ufs -o upperdir=dir1,lowerdir=dir2,workdir=wrk -f myufs dirM
```



Directories `dir1` and `dir2` are merged and that union is accessed from the UFS mount point directory `dirM`. When you create the union, directories with the same name (`dirX`) are merged. When files are encountered with the same name (for example, `dirX/file1`), only one will exist. The `workdir` option must be specified on the mount but is not actively used in the merged view. It acts as a temporary work directory for certain operations.

The directory that is specified in the mount parameter `upperdir` takes precedence over the directory that is specified in the `lowerdir` option during path name resolution. In this example, `dir1` takes precedence and you will see the version of `file1` that is in the `dir1` directory.

For more information about the mount options, see “Mounting a union file system” on page 300.

Containers and the union file system

Union file systems are used extensively by containers. With union file systems, containers can use a single file system hierarchy without having to make multiple copies, which saves on disk space. Changes that are made by one container do not affect the other containers. Other possible uses of union file systems include creating an isolated environment to test a series of file changes and manipulations without affecting the existing file system hierarchy.

Restrictions for the union file system

The following restrictions apply to the union file system.

1. Directories in an NFS file system cannot be used in the union file system mount options.
2. Directories in a union file system cannot be used in the union file system mount options.
3. The union file system cannot be exported.
4. The directory that is used in the upper directory can only be used in one union file system.
5. If any of the underlying file systems of a union file system is unmounted, it will become unusable and all operations will fail.

Mounting a union file system

These options are supported when you are mounting a union file system.

upperdir
Specifies the top-level read/write directory. Only one directory can be specified. All changes to the file system are reflected in this directory. During path name resolution, this directory is the first directory in the search order.

This option is required unless the union file system is being mounted read-only. Because directories can contain a colon in the name, the backslash (\) escape character must be used if a colon appears in one of the specified directories.

Restriction: The `upperdir` option cannot be used on multiple mounts of the union file system.

lowerdir
Specifies one or more read-only directories that are separated by colons (`:`). During path name resolution, these directories are searched after the directory that is specified in the `upperdir` option. The search order is determined by the order in which they were specified (left to right).

This option is required. Because directories can contain a colon in the name, the backslash (\) escape character must be used if a colon appears in one of the specified directories.

The `lowerdir` option can be used on multiple mounts of the union file system.

mergeufs=[on|off]
The `upperdir` and `lowerdir` options cannot specify a path that resides within another UFS file system. Specifying `mergeufs=on` will parse the merged directories from union file systems that are supplied in the lower directory option. Those directories are added to the union file system that are being mounted.

This option is optional and the default value is on.

metacopy=[on|off]
Specifies whether a change in file or directory attributes will cause an attribute-only copyup operation. When this option is off, all copyup operations are full copyup operations.

This option is optional and the default value is `off`.

redirect_dir=[on|off]
Specifies whether a rename operation of a directory in a lower directory can create a redirected directory. When this option is on, any rename of a directory with parts in a lower directory will create a directory with the new name in the upper directory. However, the content of that directory is not copied up along with the directory. Any reference to that path is redirected to the merged directory. When `redirect_dir=off` is specified, all renames of directories existing in the lower directory will fail with EXDEV and JrUFSNoRedirect.

This option is optional and the default value is on.

supercopy=[on|off]
Specifies whether a copyup of a file or directory should change the owner of the file or directory to the current user performing the copyup.

This option is optional and the default value is `off`.

workdir
Specifies a required work directory that is used internally by the union file system. This directory must be empty at mount time. It must also be on the same file system as the directory specified in the `upperdir` option.

This option is required.

Each mount option must be separated by commas. Each mount option must be followed by = and the path name of the specified directory. For example:

```
upperdir=/tmp/mydir1,lowerdir=mydir2:mydir3,workdir=/wrk
```

Behavior of a union file system

Only one directory can be specified for the `upperdir` option but more than one can be specified for the `lowerdir` option. The order in which you specify the `lowerdir` directories in the `lowerdir` option turns into the search order precedence with the `upperdir` directory always being searched first. If there are multiple nodes with the same name, the first node that is found in the search order takes precedence and makes those in the `lowerdir` directories invisible. That is, the node is *opaque*.

Merged directories

For the union file system, the root is always a merged directory that consists of the contents of the directories specified in the `upperdir` and `lowerdir` options. From the root, any directories in the `upperdir` and `lowerdir` directories with the same name will also become merged directories. When the attributes of a merged directory are retrieved, the returned attributes are those for the uppermost directory in the search order.

Copyup

By default, `upperdir` is a read/write directory. All changes that are made in the union file system are reflected in that `upperdir` directory. When changes are made to files or directories in the `lowerdir` directory, the union file system will execute a *copyup operation*, which copies the file into the `upperdir` directory. The `lowerdir` directory is always read-only and the contents are never changed. When the union file system is unmounted, the mount point directory is empty and any changes that were made are seen only in the `upperdir` directory.

The union file system has two types of copyup operations.

- A *full copyup* operation copies all of a node's data and attributes into the `upperdir` directory. The copy action occurs when a node is opened for write.
- An *attribute-only copyup* operation occurs when the attributes, such as permissions, of the node have changed. During copyup processing, all directories within the hierarchy that are not already on the `upperdir` directory are created. When a copyup operation occurs for a directory, it becomes a merged directory and the contents of the directory are not be copied.

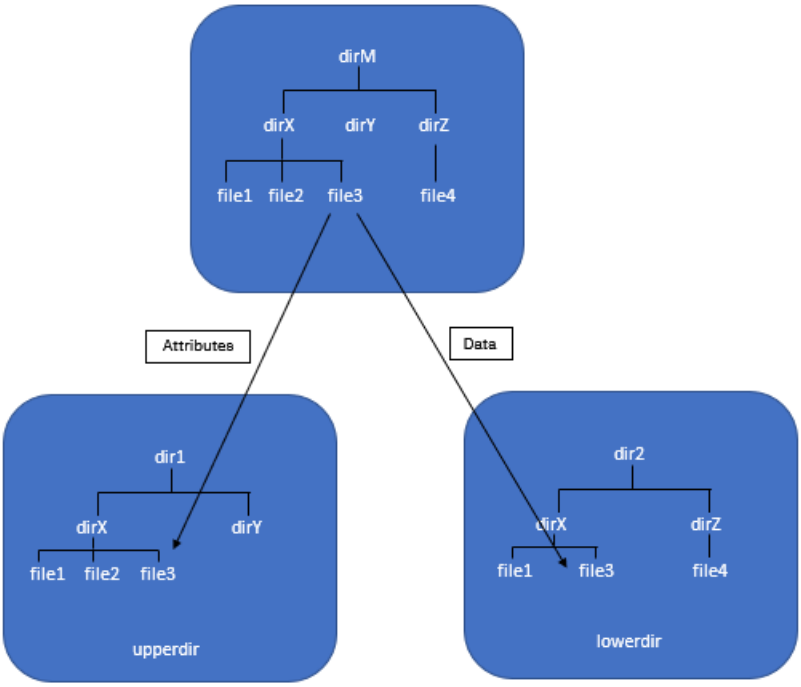
If a hard link is created to a file in a `lowerdir` directory, an attribute-only copyup operation occurs. During the copyup operation, the link to the `lowerdir` directory for a file's data is maintained. If the file is opened for write, a full copyup operation takes place, and links to the `lowerdir` directory are broken.

During copyup processing, the `ctime` of the parent directory in the `upperdir` is not preserved and will be updated.

The following examples show how the copyup operation works.

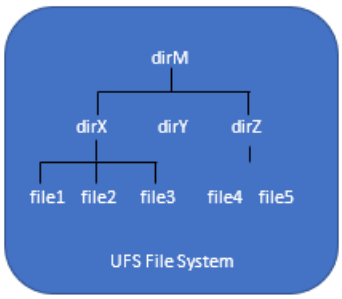
Example 1

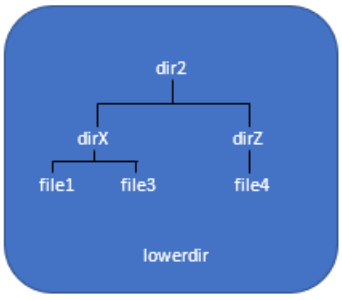
Assume that the permissions for `file3` were altered. Because `file3` resides in the `lowerdir` directory, an attribute-only copyup operation occurs, which creates an empty `file3` in the `upperdir` directory with the updated attributes. When UFS accesses the attributes of `file3`, it goes to the `upperdir` directory. However, if `file3` is read, the data is retrieved from `file3` in the `lowerdir` directory.



Example 2

This example shows what happens when a directory is copied up. Assume that `file5` is created inside of directory `dirZ`. Directory `dirZ` is copied up before `file5` is created and then becomes a merged directory. That is, the directory exists in the `upperdir` directory and `lowedir` directory and the contents are merged. As the following graphic shows, the directory was created in the `upperdir` directory but the existing file in the `lowerdir` directory was not copied up.





Whiteout nodes

When a `lowerdir` directory or previously copied-up node is deleted, a *whiteout node* is created in the `upperdir` directory. The whiteout node effectively hides the `lowerdir` directory node so that path name lookups of that name will fail. After a union file system is unmounted, the whiteout nodes remain in the `upperdir` directory. However, a subsequent union file system mount using the same `upperdir` directory will no longer treat the whiteout nodes as such but will be treated as an existing node in the directory.

Making direct actions to nodes

Making changes to union file system nodes by directly accessing the nodes in their `upperdir` or `lowerdir` directories might cause unexpected results. These results are not defined but should not cause abends.

Unmounting a union file system

The status of a union file system is not saved when it is unmounted. If the same `upperdir` and `lowerdir` options are used in a subsequent mount, then:

- Whiteout nodes are seen as any other node.
- Renamed directories from the `lowerdir` directory might be empty and does not point back to the contents on the `lowerdir` directory.
- Inode numbers are not guaranteed to be the same.

Security considerations

Node access checks are performed by the owning underlying PFS. For access checks on merged directories, only the first directory is used in the search order. If a user does not have access to the first directory in the search order, then the call will fail even if they have access to one of the other directories in the search order. Conversely, having access to the first directory will result in being granted access to view the contents of directories that would not be otherwise accessible. File access will always occur on the first instance found regardless of whether a user has access to one of the other instances in the search order.

The union file system preserves FSACCESS and FSEXEC access controls. For more information, see “Restricting access to z/OS UNIX file systems” on page 100.

Using the union file system in a shared file system

In a shared file system environment, catchup mounts are sent to nonowning systems to mount locally. Access is restricted to the owning system and accessing the union file system mount point from a nonowning system will fail with ENOSYS.

Chapter 16. Managing the PROC file system

The PROC file system contains dynamically created files that contain process and system information. Due to its nature, there are some differences between a PROC file system and a traditional file system.

- Files are not backed by DASD or storage.
- Files are not created or deleted; their existence depends on process and system parameters.
- Contents might differ between systems in a shared file system environment.

The PROC file system contains a directory for each process in the PID namespace that is associated with the file system, which makes it possible for you to obtain process information via a file system.

Creating the PROC file system

Generally, a single PROC mount is needed in a shared file system environment because under normal circumstances each PROC mount contains the same information. For more information, see “Namespace interactions” on page 305.

ServerPac installation jobs define the following FILESYSTYPE definition in SYS1.PARMLIB(BPXPRMFS):

```
FILESYSTYPE TYPE(PROC) ENTRYPOINT(BPXUPINT)
```

Edit SYS1.PARMLIB(BPXPRMFS), that is, add the mount statement to mount a file system at the `/proc` mount point. For example, you can add the following line under the FILESYSTYPE TYPE(PROC) definition:

```
MOUNT FILESYSTEM('/PROC') MOUNTPOINT('/proc') TYPE(PROC)
```

Alternatively, the mount can be created by issuing the following mount command:

```
mount –t PROC –f '/proc' /proc
```

When mounting a PROC file system:

- Each system should have the same mount statement in the BPXPRMxx parmlib member.
- Message BPXF273I is expected on IPLing systems if PROC is already mounted.
- Use the AUTOMOVE option (the default) so that the mount point is always accessible.

Namespace interactions

Every PROC mount is associated with a PID namespace, which matches the PID namespace of the process that performed the mount. If a user-initiated process did not perform the mount, then the mount is associated with the root PID namespace. This namespace association has the following effects:

- Only processes within the PID namespace have directories within the mount.
- Any displayed PIDs are in the context of this namespace, unless otherwise noted.

Contents of the PROC file system

Files and directories within the PROC file system are either process-associated or system-associated. Though the contents might change, you can find all system-associated files on any PROC mount, no matter the system. Process-associated files can be found only if the corresponding process is in the mount’s associated PID namespace. For more information about namespaces, see “PID namespaces” on page 399.

Restriction: Each z/OS system has its own PROC but PROC is not sysplex aware in the traditional sense. Although PROC is mounted at the sysplex level, you can only see data from the system that you are logged on to.

As convention, the files that are discussed in this section are discussed as if `/proc` was the mount point for the file system.

Unless otherwise mentioned, all files contain text and are read-only.

Process-associated files

Each process-associated file resides in a directory of the form `/proc/pid`, where *pid* is the PID of the process in the PID namespace that is associated with the PROC mount. The `ctime` of that directory is the start time of the process, and can be used to determine whether a PID was reused.

/proc/pid/comm
This file contains the last executed command in the process.

/proc/pid/cwd
This file is a magic symbolic link to the current working directory of the process.

/proc/pid/cmdline
This file contains the command-line arguments as seen by the process that is being executed. The contents may include binary data and null characters (NULs) if such characters were provided on exec.

/proc/pid/exe
This file is a magic symbolic link to the last executable that was executed in the process. The file may be empty if the process corresponds to an MVS program.

/proc/pid/fd/
Each file in this directory is a magic symbolic link and corresponds to an open file descriptor in the process. The link points to the file that was opened, and reading the link will produce a path to the open file. If the file was deleted, or is in some way inaccessible, reading the link will also have the text `"(deleted)"` after the path to the file.

Note: Opening these symbolic links directly opens the files without performing any access checks other than the specific file that is associated with the file descriptor, like with the `/dev/fd/xxx` character special files.

/proc/pid/mountinfo
This file contains a list of the mounts and detailed mount information in the process's mount namespace that are accessible from the process's root directory. If a mount is added or removed while the file is being read, there is no guarantee that all mounts will be shown. The following is an example line:

```
6 5 0:6 / /my/mount rw,nosuid unbindable - TFS MYMOUNT rw,-s 10
```

The whitespace-delimited items are, in order:

1. The device number of the mount.
2. The device number of the mount's parent.
3. The prefix "0:" followed by the device number of the real file system for the mount. In most cases, this is identical to the first field but may be different in the case of a bind mount.
4. The path to root vnode with respect to the root of the real file system. For normal mounts this will be "/" but can be different in the case of a bind mount.
5. The mount point, with respect to the root of the process.
6. LFS-related mount options.
7. Additional LFS-related mount options. Currently "unbindable" is the only option here which might be present. This list is terminated by the dash ("-").
8. The file system type.
9. The file system name.
10. The file system parameters.

/proc/pid/mounts
This file contains a list of mounts in the process's mount namespace that are accessible from the process's root directory. If a mount is added or removed while the file is being read, there is no guarantee that all mounts will be shown. The following is an example line:

```
MYMOUNT /my/mount TFS rw,nosuid,-s 10 0 0
```

The white space-delimited items are, in order:

1. The file system name.
2. The mount point.
3. The file system type.
4. The LFS-related mount options and mount parameters of the file system.
5. Two 0's, present for compatibility.

/proc/pid/mountstats
This file contains a more human-readable list of mounts in the process's mount namespace that are accessible from the process's root directory. If a mount is added or removed while the file is being read, there is no guarantee that all mounts will be shown. All lines have the following format:

```
device filesysname mounted on mountpoint with fstype filesystype
```

Where *filesysname* is the name of the file system, *mountpoint* is the mount point of the file system, and *filesystype* is the file system type.

/proc/pid/ns/
A directory containing magic symbolic links pointing to files that represent the namespaces the process belongs to. While these files can be opened, their intent is to be passed to namespace-related syscalls to manage the changing of namespaces.

/proc/pid/root
This file is a magic symbolic link to the process's root directory.

/proc/pid/stat
A file containing various process-related statistics. The white space-delimited values represent the following, in order:

1. The PID of the process in the namespace associated with the PROC mount.
2. The name of the executable, in parentheses.
3. The status of the process:
   - R – Running
   - S – Sleeping
   - T – Stopped
   - Z – Zombie
   - X – Dead/Terminating
4. The PID of the parent process.
5. The process group.
6. The session ID.
7. 0 – Reserved for compatibility.
8. 0 – Reserved for compatibility.
9. 0 – Reserved for compatibility.
10. 0 – Reserved for compatibility.
11. 0 – Reserved for compatibility.
12. 0 – Reserved for compatibility.
13. 0 – Reserved for compatibility.

14. User usage time, in milliseconds.
15. System usage time, in milliseconds.
16. Cumulative child user usage time, in milliseconds.
17. Cumulative child system usage time, in milliseconds.
18. Nice value.
19. Process priority.
20. The number of process threads.
21. 0 – Reserved for compatibility.
22. Process start time, in seconds since epoch.
23. Virtual storage size, in bytes.
24. Real storage size, in bytes.
25. Real storage limit, in bytes.
26. 0 – Reserved for compatibility.
27. 0 – Reserved for compatibility.
28. 0 – Reserved for compatibility.
29. 0 – Reserved for compatibility.
30. 0 – Reserved for compatibility.
31. 0 – Reserved for compatibility.
32. 0 – Reserved for compatibility.
33. 0 – Reserved for compatibility.
34. 0 – Reserved for compatibility.
35. 0 – Reserved for compatibility.
36. 0 – Reserved for compatibility.
37. 0 – Reserved for compatibility.
38. 0 – Reserved for compatibility.
39. 0 – Reserved for compatibility.
40. 0 – Reserved for compatibility.
41. 0 – Reserved for compatibility.
42. 0 – Reserved for compatibility.
43. 0 – Reserved for compatibility.
44. 0 – Reserved for compatibility.
45. 0 – Reserved for compatibility.
46. 0 – Reserved for compatibility.
47. 0 – Reserved for compatibility.
48. 0 – Reserved for compatibility.
49. 0 – Reserved for compatibility.
50. 0 – Reserved for compatibility.
51. 0 – Reserved for compatibility.
52. Exit code, if the process has exited.

/proc/pid/status

This file contains process-related statistics with line containing a white-space delimited field name and value. Each field is described in the following table:

Name	Description
Name	The executable name.
Umask	The umask for the process.
State	The process's running state.
Pid	The PID of the process in the PID namespace associated with the mount.
PPid	The PID of the parent process.
Uid	The real, effective, saved, and effective (for compatibility) UIDs for the process.
Gid	The real, effective, saved, and effective (for compatibility) GIDs for the process.
FDsize	The maximum number of file descriptors the process can open.
NSpid	The PIDs for each of the PID namespace that the process is in, in order from the PID namespace associated with the mount down to the namespace of the process.
NSpgid	The process group ID for each of the PID namespaces the process is in, in order from the PID namespace associated with the mount down to the namespace of the process.
Threads	The number of threads in the process.
SigPnd	The signals pending in the initial process thread.
ShdPnd	The signals pending among all threads in the process.
SigBlk	The current blocked signals mask.
SigIgn	Ignored signals - (not by default).
SigCgt	Signals currently caught.
NoNewPrivs	1 if **NoNewPrivs** is enabled, 0 otherwise.
VmSize	The virtual storage size.

Each of the signal values is hexadecimal values and are written with the rightmost bit representing bit 0, for compatibility reasons.

System-associated files

The following files are found on all systems.

`/proc/filesystems`
This file contains a list of all the mountable file systems.

`/proc/self`
This file is a symbolic link containing the current process's PID in the PID namespace associated with the PROC mount. If the current process is not in that namespace, reading the link will result in an error.

`/proc/mounts`
This file is a symbolic link to `self/mounts`.

`/proc/stat`
A file that contains the following fields. Each line in the file consists of a field name followed by its value:

Name	Description
btime	The IPL time, in seconds since epoch.
processes	The number of processes.
procs_running	The number of processes in the running state.

/proc/sys/fs/epoll/max_user_watches
This file contains the maximum number of file descriptors that a user can watch with epoll.

/proc/sys/fs/inotify/max_queued_events
This file contains the maximum number of events that can be queued with inotify.

/proc/sys/fs/inotify/max_user_instances
This file contains the maximum number of inotify instances that a user can create.

/proc/sys/fs/inotify/max_user_watches
This file contains the maximum number of file descriptors that a user can watch with inotify.

/proc/sys/kernel/hostname
This file contains the system name.

/proc/sys/kernel/pid_max
This file contains the maximum value that a PID can have.

/proc/sys/uptime
The amount of time since system IPL, in seconds since epoch.

/proc/sys/version
This file contains the z/OS product name, version, and FMID.

Security considerations

The following factors determine file ownership:

- For process-associated files, the UID and GID of the file match the UID and GID of the process. If the process was made not dumpable, they will be UID=0 and GID=0.
- For system-associated files, the UID and GID of the file match the UID and GID of the process that mounted the PROC file system.

Some process-associated symbolic links in the PROC file system are known as *magic symbolic links*. They have special rules about whether a user can read the symbolic link or resolve a path using the link. Those rules follow an ordered set of checks to determine whether access is either allowed or denied and are as follows:

1. If the targeted process matches the current process, access is allowed.
2. If the current user is a superuser, access is allowed.
3. If the targeted process is not dumpable, access is denied.
4. If the current process's effective UID and effective GID match the targeted process's UID and GID in the real, effective, and saved sets, access is allowed.
5. Otherwise, access is denied.

Process directories can only be found if the user owns the process or has authority to the SUPERUSER.PROCESS.GETPSENT resource profile in the UNIXPRIV class.

Chapter 17. Setting up for daemons

A *daemon process* is a process that runs in the background and is not associated with any particular terminal or user. Daemons have superuser authority and can issue authorized functions such as setuid(), seteuid() and spawn() to change the identity of a user's process. When setting up daemons, security levels need to be taken into consideration.

Lists of subtasks

Subtasks	Associated procedure
Establishing the correct level of security for daemon	"Steps for preparing the security program for daemons" on page 314
Customizing the system for IBM-supplied daemons	"Steps for defining programs from load libraries to program control" on page 316 "Steps for checking UNIX files for program control " on page 317 "Steps for setting up enhanced program security" on page 320 "Steps for customizing the cron daemon" on page 325
Customizing the system for IP-supplied daemons	"Steps for customizing the system for IP-supplied daemons" on page 321
Customizing the IBM-supplied daemons	"Steps for customizing the inetd daemon" on page 322 "Steps for customizing the uucpd daemon" on page 323
Setting up security procedures for daemons	"Steps for setting up security procedures for daemons" on page 330
Tracking down problems when setting up daemons and servers	"Steps for finding modules that were not defined to program control" on page 335
Setting up for rlogin	"Steps for setting up for rlogin" on page 338

If you require a high level of security in your z/OS system and do not want superusers to have access to z/OS resources such as SYS1.PROCLIB, read the following sections:

- "Comparing UNIX security and z/OS UNIX security" on page 311.
- "Establishing the correct level of security for daemons" on page 313.

Comparing UNIX security and z/OS UNIX security

Some of the people who perform z/OS UNIX tasks have a background in MVS, while others have experience in UNIX systems other than z/OS UNIX.

MVS, traditional UNIX, and z/OS UNIX systems manage user identities differently. Table 37 on page 312 contrasts various aspects of security on these systems.

Table 37. Comparing traditional UNIX, MVS, and z/OS UNIX security. Security on traditional UNIX, MVS, and z/OS UNIX systems is compared.

Category	Traditional UNIX	MVS	z/OS UNIX
User identity	Users are assigned a unique UID, a 4-byte integer, and user name.	Users are assigned a unique user ID of 1-to-8 characters.	Users are assigned a unique user ID with an associated UID.
Security identity	UID	User ID	UID for accessing traditional UNIX resources and the user ID for accessing traditional z/OS resources
Login ID	Name that is used to locate a UID	Same as the user ID	Same as the user ID
Special user	Multiple user IDs can be assigned a UID of 0.	RACF administrator assigns necessary authority to users.	Multiple user IDs can be assigned a UID of 0 or users can be permitted to BPX.SUPERUSER.
Data set access	Superuser can access all files.	All data sets controlled by RACF profiles.	Superuser can access all UNIX files; data sets that are controlled by RACF profiles.
Identity change from superuser to regular user	Superuser can change the UID of a process to any UID that is using setuid() or seteuid() functions.	APF-authorized program can invoke SAF service to change identity.	There are two options. If BPX.DAEMON is not defined, the superuser can change the UID of a process to any UID using setuid() or seteuid() functions. Or, the superuser must be permitted to BPX.DAEMON in order to change UIDs.
Identity change from regular user to superuser	The su shell command allows change if user provides password for the root. Password phrases are not used in traditional UNIX security.	No provision for unauthorized user to change identity.	The su shell command allows change if the user is permitted to BPX.SUPERUSER or if the user provides the password or password phrase of a user with a UID of 0.
Identity change of a regular user from one UID to another UID	The su shell command allows change if user provides password. Password phrases are not used in traditional UNIX security.	No provision for unauthorized user to change identity.	The su shell command allows change if user provides password or password phrase.
Terminate user processes	Superuser can kill any process.	MVS operator can cancel any address space.	Superuser can kill any process.

Table 37. Comparing traditional UNIX, MVS, and z/OS UNIX security. Security on traditional UNIX, MVS, and z/OS UNIX systems is compared. *(continued)*			
Category	**Traditional UNIX**	**MVS**	**z/OS UNIX**
Multiple logins	Users can login to a single user ID multiple times.	Users can only log on to TSO/E once per user ID.	Users can rlogin multiple times to a single user ID and logon once to TSO/E at the same time.
Login daemons	`inetd`, `rlogind`, `lm`, and `telnetd` process user requests for login. A process is created with the user identity (UID).	TCAS and VTAM process user requests for logon. A TSO/E address space (process) is created with the user identity (user ID).	Users can log on to TSO/E or login by using one of the login daemons. In all cases, an address space is created with both an MVS identity (user ID) and a UID.

Establishing the correct level of security for daemons

Kernel services support two levels of appropriate privileges: UNIX level and z/OS UNIX level. This lets you distinguish superusers from daemons. You need to determine which level of security is appropriate for your installation.

UNIX level

If the BPX.DAEMON resource in the FACILITY class is not defined, your system has UNIX-level security. In this case, the system is less secure.

This level of security is for installations where superuser authority has been granted to system programmers. These individuals already have permission to access critical data sets such as PARMLIB, PROCLIB, and LINKLIB. These system programmers have total authority over a system.

Programs that run with superuser authority have daemon level authority. They can issue MVS identity-changing services such as setuid(), seteuid(), and __spawn() without having first issued a successful _passwd() for the target user ID.

To use the UNIX level of security, assign UID(0) to the superuser. Also, assign UID(0) to the user ID that is used to run daemon programs; for example, inetd or cron.

RACF with enhanced program security, BPX.DAEMON, and BPX.MAINCHECK

If you enable enhanced program security, and you have any daemons or servers that run execute-controlled programs (MVS programs defined to RACF in the PROGRAM class using EXECUTE authority, or loaded from libraries using EXECUTE authority), then you must define the initial program executed by your daemon or server as a trusted ("MAIN") program to RACF via the PROGRAM class. If this initial program resides in the z/OS UNIX file system, rather than in an MVS library, you will need to move it to an MVS library.

Additionally, you can choose whether to extend the enhanced program security protection to your UNIX daemons and servers that do not make use of RACF execute-controlled programs. Enable this function by defining the profile BPX.MAINCHECK to RACF in the FACILITY class. Again, need to ensure that the initial program executed by your daemon or server resides in an MVS library and you need to define it to RACF as a PROGRAM with the MAIN attribute.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to BPX.DAEMON FACILITY. You will also have to choose the

users to whom you give the OMVS security profile segments. To accomplish this, see “Steps for preparing the security program for daemons” on page 314.

“Steps for setting up enhanced program security” on page 320 explains how to set up enhanced program security.

BPX.DAEMON

If the BPX.DAEMON resource in the FACILITY class is defined, your system has z/OS UNIX security. Your system can exercise more control over your superusers.

This level of security is for customers with stricter security requirements who need to have some superusers maintaining the file system but want to have greater control over the z/OS resources that these users can access. Although BPX.DAEMON provides some additional control over the capabilities of a superuser, a superuser should still be regarded as a privileged user because of the full range of privileges the superuser is granted.

The additional control that BPX.DAEMON provides involves the use of kernel services such as setuid() that change a caller's z/OS user identity. Any user can issue a setuid() which follows a successful __passwd() call to the same target user ID. However, a user with daemon authority can issue setuid() without knowing the target user's password or password phrase. With BPX.DAEMON defined, a superuser process can run these types of change services and identity if the following statements are true:

- The caller's user identity was permitted to BPX.DAEMON.
- All programs running in the address space have been loaded from a library that is controlled by a security product. A library that is identified to RACF program control is an example. You can identify individual files as controlled programs. For more information, “Customizing the system for IBM-supplied daemons” on page 315.

  Programs that were loaded from MVS libraries do not need to be controlled programs if BPX.DAEMON.HFSCTL has been set up. Only UNIX files are checked for program control. For information about setting up BPX.DAEMON.HFSCTL, see “Checking UNIX files for program control” on page 317.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you must choose which daemons to permit to BPX.DAEMON. You will also have to choose the users to whom you give the OMVS security profile segments. To accomplish this, see “Steps for preparing the security program for daemons” on page 314.

The RACF WARN mode is not supported for BPX.DAEMON.

Steps for preparing the security program for daemons

Before you begin: You need to follow the procedures for security as described in:

1. “Preparing RACF” on page 46 and “Steps for preparing RACF ” on page 46.
2. “Defining z/OS UNIX users to RACF” on page 51 and “Defining group identifiers (GIDs)” on page 57.
3. “Controlling access to files and directories” on page 83.

Perform the following steps to prepare RACF for daemons:

1. Define BPX.DAEMON to permit users that are known as daemons to query or modify the z/OS security environment of a process. You must use the name BPX.DAEMON. Substitutions are not allowed

   ```
   RDEFINE FACILITY BPX.DAEMON UACC(NONE)
   ```

   The system administrator must be defined to the daemon FACILITY class so that if a daemon process fails, the system administrator can restart it. To authorize a current RACF security administrator to be a superuser who can restart daemons, issue:

   ```
   PERMIT BPX.DAEMON CLASS(FACILITY) ID(RACFADM) ACCESS(READ)
   ```

2. If this is the first FACILITY class that the installation has defined, activate it.

```
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY)
```

3. Give daemon authority to the kernel. Most daemons that inherit their identities from the kernel address space are started from `/etc/rc`. Be sure that you know which ID was already permitted to BPX.DAEMON, as part of the security setup described in “Steps for preparing RACF ” on page 46. In this example, the ID is OMVSKERN.

   Example: To authorize the OMVSKERN user ID, issue:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

4. Define a superuser with a user ID of BPXROOT on all systems so that daemon processes can invoke setuid() for superusers.

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0)
      HOME('/') PROGRAM('/bin/sh'))
NOPASSWORD
```

   The NOPASSWORD option indicates that BPXROOT is a protected user ID that cannot be used to enter the system by using a password or password phrase. The user ID will not be revoked due to invalid logon attempts.

5. On the SUPERUSER statement in BPXPRMxx, specify the user ID that the kernel will use when you need a user ID for UID(0).

   Example:

```
SUPERUSER(BPXROOT)
```

   If you do not specify the SUPERUSER statement, the default is BPXROOT.

   The BPXROOT user ID should not be permitted to the BPX.DAEMON FACILITY class profile. The BPXROOT user ID is used when a daemon process invokes setuid() to change the UID to 0 and the user name has not been previously identified by getpwnam() or by the _passwd() function. This action prevents the granting of daemon authority to a superuser who is not defined to BPX.DAEMON.

When you are done, you have prepared RACF for daemons. To complete the security setup, you must also activate program control, as described in “Customizing the system for IBM-supplied daemons” on page 315.

Customizing the system for IBM-supplied daemons

z/OS UNIX supplies these daemons:

- `inetd`—the network daemon
- `rlogind`—the remote login daemon
- `cron`—the clock daemon
- `uucpd`—the UUCP daemon

The `syslogd` daemon, which is used to route messages, is shipped with TCP/IP and is documented in their library.

If you are defining BPX.DAEMON for a higher level of security, you need to customize the system for IBM-supplied daemons. Many daemons require BPX.DAEMON authority and must have all modules loaded in their address spaces identified as being defined to program control.

For more information, see Protecting programs in *z/OS Security Server RACF Security Administrator's Guide*.

Defining modules to program control

In most cases, programs loaded into an address space that requires daemon authority must be controlled programs. All programs must be program controlled. However, programs loaded from MVS libraries do not have to be program controlled if BPX.DAEMON.HFSCTL has been set up. (See "Checking UNIX files for program control" on page 317.) In that case, only UNIX files are checked for program control.

If a program that is not a controlled program is loaded, the address space is marked dirty and cannot perform daemon activities. For more information about dirty address spaces, see "Handling dirty address spaces" on page 319.

Steps for defining programs from load libraries to program control

Before you begin, you need to know which programs you want to define to program control. If you run with enhanced program security, you might need to define some programs with the MAIN attribute via the APPLDATA operand on the PROGRAM profile.

Perform the following steps to define programs from traditional load libraries to program control.

1. Activate the RACF program control (both access control to load modules and program access to data sets).

   ```
   SETROPTS WHEN(PROGRAM)
   ```

   _____

2. Define one of the following profiles.

   a. For a particular program, define a discrete RACF PROGRAM class profile:

      ```
      RDEFINE PROGRAM membername ADDMEM('datasetname'/volser/NOPADCHK) UACC(READ)
      ```

   b. For all members in a data set:

      ```
      RDEFINE PROGRAM * ADDMEM('datasetname'/volser/NOPADCHK) UACC(READ)
      ```

   _____

3. Refresh the in-storage copy of the PROGRAM profile.

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

_____

When you are done, you have defined a program from a load library to program control.

Tips:

1. PROGRAM profile * provides the same function as PROGRAM profile **. If you already have a PROGRAM profile * defined, do not create an ** profile. Instead, issue the RALTER command against PROGRAM * with the same operands shown in the RDEFINE PROGRAM example.
2. If you are running in a sysplex with a shared RACF data base and your system libraries are also shared, then leaving the VOLSER off will allow you to use the same RACF definitions on all systems in the sysplex.
3. Any time you add, change, or delete a profile in the PROGRAM class (with RDEFINE, RALTER, PERMIT, or RDELETE), you must update the in-storage copy of the PROGRAM profile.

```
SETROPTS WHEN(PROGRAM)
REFRESH
```

4. Daemons that are shipped by z/OS reside in the file system and are controlled programs, so you do not need to define them to program control. For example, suppose you have a daemon named `server1`. The file `/bin/server1` would have the sticky bit on. Member SERVER1 would reside in SYS1.LINKLIB and be defined as a controlled program.

   ```
   RDEFINE PROGRAM SERVER1
   ADDMEM('SYS1.LINKLIB'/'******'/NOPADCHK) UACC(READ)
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

   You do not need to define the daemons that are shipped by z/OS if you decide to define BPX.MAINCHECK, as discussed in “Using enhanced program security” on page 320.

5. Daemons can load locales from the file system or from MVS load modules. If they are loaded from MVS load libraries, then these modules must be marked program-controlled. If they are loaded from the file system, the program control extended attribute bit must be set. The locales shipped by IBM already have this extended attribute bit set.

Defining programs in UNIX files to program control

Before you can define programs in UNIX files to program control, you need READ access to the BPX.FILEATTR.PROGCTL resource in the FACILITY class. Then use the `extattr` command with the +p option to set the program control extended attribute.

For example, to set the program control extended attribute in the file named `proga`, issue:

```
extattr +p /user/sbin/proga
```

The attribute is turned off if there is any activity that can change the contents of the file. If this happens, a system programmer with the appropriate privilege will have to verify that the file is still correct. Then the programmer will have to issue the `extattr` command to set the program control attribute back on. To find out if the program control extended attribute has been set, use the `ls -E` command.

Using sanction lists

You can compile a list to contain the lists of path names and program names that are sanctioned by the installation for use by program-controlled programs. This file contains properly constructed path names and program names as described in Path and path name in *z/OS UNIX System Services User's Guide*. For more information, see “Using sanction lists” on page 95.

Checking UNIX files for program control

If you want only UNIX files to be checked for program control, and do not want programs loaded from MVS libraries to be checked, you can set up BPX.DAEMON.HFSCTL. However, doing this weakens some of the security provided by the BPX.DAEMON resource. It should be done only in restricted and carefully considered cases, or if you do not already run with BPX.DAEMON but want to gain only a subset of the benefits of running with BPX.DAEMON.

Steps for checking UNIX files for program control

Before you begin, you need to know whether BPX.DAEMON has already been activated.

Perform the following steps to set up the BPX.DAEMON.HFSCTL resource in the FACILITY class.

1. Activate BPX.DAEMON, if it is not already active. For more information about that topic, see “Steps for preparing the security program for daemons” on page 314.

   _____

2. Define the resource profile.

```
RDEFINE FACILITY BPX.DAEMON.HFSCTL UACC(NONE)
```

_____

3. Give READ access to users.

```
PERMIT BPX.DAEMON.HFSCTL CLASS(FACILITY) ID(uuuuuu) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

_____

When you are done, you have set up the BPX.DAEMON.HFSCTL resource in the FACILITY class.

Defining UNIX files as APF-authorized programs

The authorized program facility (APF) allows your installation to identify system or user programs that can use sensitive system functions. To be APF-authorized, programs must reside in APF-authorized libraries, and be link-edited with authorization code AC=1. The program must also be the initial program (that is, it must be the job step task program), or it was invoked by a caller that is running APF-authorized.

If the specified program is going to be invoked as a job step program, you must link-edit it with AC=1. For example:

```
c89 -Wl, AC=1
```

To avoid possible integrity problems, do not set AC=1 if the program will be run in an APF-authorized environment but not as the job step program (such as DLL).

The APF rules for programs that reside in the z/OS UNIX file system are similar to those for programs that reside in authorized libraries. Setting the APF-authorized extended attribute bit should be thought of as putting that program into an authorized library. If you try to run a program from an authorized library that is not linked AC=1, it will not run APF-authorized, but that same program could be fetched by another that is running APF-authorized and executed in the authorization state in which it is called, or even have its state changed.

Tip: To find out whether the APF-authorized extended attribute of the UNIX file was set, use the `ls -E` command.

Compiling a list of sanctioned path names and program names

You can compile a list to contain the lists of path names and program names that are sanctioned by the installation for use by APF-authorized programs. This file contains properly constructed path names and program names. For information about rules for path names, see Path and path name in *z/OS UNIX System Services User's Guide*. For more information about sanction lists, see “Using sanction lists” on page 95.

Controlling who can set the APF-authorized attribute

Use the BPX.FILEATTR.APF resource in the FACILITY class to control which users are allowed to set the APF-authorized attribute in a z/OS UNIX file.

The following example shows the RACF command that is used to give the necessary permission to user Ralph Smorg with user ID SMORG:

```
RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
PERMIT BPX.FILEATTR.APF CLASS(FACILITY) ID(SMORG) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

To set the APF-authorized extended attribute in an executable file, issue the `extattr` command with the `+a` option. In the following example, `proga` is the name of the file.

```
extattr +a /user/sbin/proga
```

Defining UNIX files as shared library programs

Shared libraries are programs that, when loaded, are put in the shared library region for system-wide sharing. A program is loaded as a shared library program if the executable file has the shared library extended attribute set.

To find out if the shared library extended attribute has been set, use the `ls -E` command.

Setting the shared library attribute

The BPX.FILEATTR.SHARELIB resource in the FACILITY class controls who can set the shared library extended attribute. You need to have at least READ access before you can set the shared library extended attribute.

The following example shows the RACF command that was used to give READ access to user Ralph Smorg with user ID SMORG:

```
RDEFINE FACILITY BPX.FILEATTR.SHARELIB UACC(NONE)
PERMIT BPX.FILEATTR.SHARELIB CLASS(FACILITY) ID(SMORG) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

To set the shared library attribute, issue the `extattr` command with the `+l` option.

In the following example, `progdll` is the name of the file.

```
extattr +l  /user/sbin/progdll
```

Note: Once the ST_SHARELIB bit has been set for a module, any program, whether or not it has read access to the BPX.FILEATTR.SHARELIB FACILITY, will be able to load modules into the system shared library and use those that are already loaded.

Handling dirty address spaces

A dirty address space is an address space requiring daemon authority that has had an uncontrolled program loaded into it. Dirty address spaces, which are also known as *dirty environments*, cannot perform daemon activities.

If the BPX.DAEMON resource in the FACILITY class has been defined, then programs that are loaded from MVS libraries are checked for program control. The checking is bypassed only if BPX.DAEMON.HFSCTL is defined and the user is permitted to it.

Programs in files are controlled programs if they have the program control attribute set. If a program that is not a controlled program is loaded, the address space is marked dirty and cannot perform daemon activities. If an address space was marked dirty, you can load a controlled program but it will not be able to do any controlled functions such as setuid(). All BPX.SERVER and BPX.DAEMON privileges are revoked, including the right to check passwords and password phrases.

Programs can be defined to program control in the following ways:

- The load modules can be loaded from a load library, where all modules in the library can be defined to program control, or specific modules in the library can be defined to program control.
- The module can reside in the file system with the sticky bit on. The system then searches the MVS search order and the rules for program control apply.
- The module can reside in the file system with the external attribute set for program control.

RACF supports program control. Other security products might not. If you are using a security product that does not support program control, you might still have BPX.DAEMON defined. In this case, the only situation that will mark an address space dirty is a load from the file system where the program is not defined to program control.

Using enhanced program security

If you choose to use the enhanced program security function, and you have daemons that use programs defined to RACF as execute-controlled programs (by having EXECUTE access to the RACF PROGRAM profile that defines the program, or EXECUTE access to the library containing the program), then you need to take some special actions to configure your daemons so that they will run properly.

In an environment with enhanced program security, and using execute-controlled programs, the initial program executed by a daemon must be defined to RACF with a profile in the PROGRAM class, and that profile must specify the MAIN option via the profile's APPLDATA. However, only programs loaded from an MVS library can be defined using the RACF PROGRAM class; you cannot define programs loaded from the z/OS UNIX file system. Therefore, if you have daemons that use execute-controlled programs, you need to move their initial program from the z/OS UNIX file system into an MVS library so that you can define it completely to RACF.

Additionally, if you run with enhanced program security and have the BPX.DAEMON FACILITY class profile defined, you can use another FACILITY profile to request that z/OS UNIX apply tighter security controls to your daemons. Typically, with BPX.DAEMON defined, z/OS UNIX will work with RACF to enforce a clean environment for any daemon. In this case, the daemon can run only those programs defined to the RACF PROGRAM class or marked controlled via the `extattr` shell command with the +p option.

For additional security, you can define FACILITY profile BPX.MAINCHECK. When you do that, z/OS UNIX and RACF will require that the first program your daemon executes must be defined to RACF using a PROGRAM profile with the MAIN option for use of execute-controlled programs. If you define BPX.MAINCHECK, then you need to move the first program that any daemon executes from to an MVS library if it currently resides in the UNIX file system.

Steps for setting up enhanced program security

Before you begin, you need to have:

1. RACF set up as your security product.
2. Enabled RACF enhanced program security.
3. Enabled BPX.MAINCHECK.
4. Determined which privileged programs you run that are affected by setting up RACF enhanced program security. The RACF programs that would be affected are the main job-step programs of one of the following types of privileged applications:
   - z/OS UNIX applications that require a program-controlled environment. This includes applications that require permission to BPX.DAEMON, BPX.SERVER, or BPX.SRV.*userid* or those that use a privileged function like __passwd(). Examples of applications that would be affected by this are rlogin, telnet and su.
   - Applications that gain access to MVS data sets by using RACF program access to data sets (PADS) via entries in a DATASET profile's conditional access list.

Perform the following steps to set up ENHANCED program security mode.

1. Turn on RACF ENHANCED program security mode. For more information about ENHANCED program security mode, see Program security modes in *z/OS Security Server RACF Security Administrator's Guide*.

   _____

2. Ensure that all affected MAIN job-step programs are in an MVS load library in your MVS load library search order. They should have either the sticky bit attribute turned on (see “Verifying that the sticky bit is on” on page 333) or have been set up as an external link z/OS UNIX file (see “Using external links to access MVS load libraries” on page 334).

If you use the warning mode provided by RACF enhanced program security as a way to determine which programs will be affected by the new enhanced security checking, note that in warning mode, the applications will not fail but you will get messages that indicate which programs are affected

_____

3. Define the BPX.MAINCHECK security profile.

```
RDEFINE FACILITY BPX.MAINCHECK UACC(NONE)
```

_____

4. Re-IPL.

_____

When you are done, you have set up enhanced program security.

Tip:

1. You can partially activate enhanced program security by defining the profile before restarting OMVS or issuing a SET OMVS or SETOMVS command. However, only address spaces that are started after enhanced program security was enabled are affected. Use this partial enablement for testing purposes only.
2. Because the new RACF enhanced security checking requires a completely controlled program environment, testing using dbx might be restricted because it can cause the program environment to be considered uncontrolled. Testing a trusted MAIN program under dbx might require that the RACF enhanced security checking be set up in warning mode or that BPX.MAINCHECK be undefined. Attempting to do otherwise might cause some privileged operations to fail while under dbx control.

Remain in warning mode until you have done at least one IPL, to ensure that you have tested with all your daemons.

Customizing the system for IP-supplied daemons

The `syslogd` daemon, which is used to route messages, is shipped with z/OS Communications Server, (TCP/IP Services). Other daemons provided by z/OS Communications Server are `otelnetd` and `orexecd`.

Before you can use the daemons, you have to permit each daemon to the BPX.DAEMON FACILITY class profile and then ensure that the library that contains the daemon is added to the program control profile.

Steps for customizing the system for IP-supplied daemons

Before you begin, you need to know what IP-supplied daemons you will be using.

Perform the following steps to customize the system for IP-supplied daemons.

1. Permit each daemon to BPX.DAEMON.

   For example, set up the `syslogd` daemon, which is in the SEZALOAD library:

   ```
   PERMIT BPX.DAEMON CLASS(FACILITY) ID(syslogd)  ACCESS(READ)
   ```

   _____

2. Add the library that contains the library to the program control profile.

   For example, define the SEZALOAD library to the PROGAM class:

   ```
   RALT PROGRAM * ADDMEM('tcpip.SEZALOAD'/'volser'/NOPADCHCK) UACC(READ)
   ```

   In that example, you set up the `syslogd` daemon and then add the SEZALOAD library to the program control profile. You can start using that daemon.

See “Customizing the system for IBM-supplied daemons” on page 315 for more information about program control. *z/OS Communications Server: IP Configuration Guide* also has more information.

When you are done, you have customized the system for IP-supplied daemons.

Customizing the IBM-supplied daemons

This section discusses customizing these IBM-supplied daemons: `inetd`, `uucpd`, `rlogind`, and `cron`.

Customizing the inetd daemon

The `inetd` daemon provides service management for a network. It reduces system load by invoking other daemons only when they are needed and by providing several simple Internet services internally without invoking other daemons.

After it has been started, the `inetd` daemon monitors network sockets for services that are listed in `/etc/inetd.conf`. That file tells the `inetd` daemon which services to support and how to handle service requests. When `inetd` receives a request on one of these sockets, it determines which service corresponds to that socket. Then it either handles the service request itself or invokes the appropriate server. For more information about the `/etc/inetd.conf` file, see the description of inetd daemon - Provide service management for networks in *z/OS UNIX System Services Command Reference*.

For z/OS UNIX, `inetd` handles rlogin, telnet, rsch, rexec, and others. It uses a configuration file in `/etc/inetd.conf` when it handles the requests.

Warning: The rexec, rlogin, rshd, telnet, and uucp services are by nature unsecure because they require user authentication that uses cleartext sockets. A remote attacker can exploit this vulnerability to sniff logins and passwords. Avoid these services if possible.

Steps for customizing the inetd daemon

Before you begin: TCP/IP must be properly configured and started.

Perform the following steps to customize the inetd daemon.

1. Copy `/samples/inetd.conf` to `/etc/inetd.conf`.

   ---

2. Decide which services you want to support, such as rlogin and telnet. There is no list of daemons that can be started from `inetd`. To find out whether a daemon can run under `inetd`, check its documentation. The documentation should also tell you what its `inetd.conf` entry should look like.

   ---

3. Decide on a user name for the services. You can use the one in the sample `inetd.conf` (OMVSKERN). You can also use a different user name for each service. Some daemons might not require as many privileges as others.

   ---

4. Set up the user names in RACF, with appropriate privileges. You should consider whether to use BPX.DAEMON support. (For more information, see “Establishing the correct level of security for daemons” on page 313.)

   For a multilevel secure environment: If the SAF FACILITY class resource profile BPX.POE is defined, you must grant the user ID assigned to INETD to at least READ access to this profile. For example:

   ```
   PERMIT BPX.POE CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

   ---

5. Uncomment or add a line in `inetd.conf` for each service that you want to support. Make any changes needed to the lines for supported services. For the syntax of `inetd.conf` entries, see inetd daemon - Provide service management for networks in *z/OS UNIX System Services Command Reference*. Also see the appropriate documentation for the various daemon programs for the requirements for each daemon.

   _____

6. Make sure that each service is listed in TCPIP.ETC.SERVICES or `/etc/services` with the appropriate port number.

   _____

7. Arrange for `inetd` to be started on each IPL. The most common way to do this is to start it from `/etc/rc`. It can also be started from a started task using BPXBATCH with PARM='SH...' or from a shell session of a user with appropriate authority.

   If you start `inetd` from `/etc/rc`, then messages will be sent to `/etc/log`. If you start `inetd` as a started task and you have syslogd running, then any `inetd` messages will go to syslogd. You can have syslogd direct those messages to the MVS console with a statement like the following in your `/etc/syslog.conf` file:

   ```
   *.INETD*.daemon.debug      /dev/console
   ```

   _____

When you are done, you have customized the `inetd` daemon.

Customizing the uucpd daemon

The `uucpd` daemon handles the communications between local and remote sites for file transfer via TCP/IP connections in an UUCP network. For more information, see Chapter 10, “Configuring the UNIX-to-UNIX copy program (UUCP),” on page 227 and the descriptions of the various `uucp` commands in *z/OS UNIX System Services Command Reference*.

Warning: The uucpd daemon is by nature unsecure because it requires user authentication that uses cleartext sockets. A remote attacker can exploit this vulnerability to sniff logins and passwords. Do not use this daemon if possible.

Steps for customizing the uucpd daemon

Before you begin, you need to set up the `uucpd` daemon to `/etc/inetd.conf`.

For example, add the following lines to `/etc/inetd.conf`:

```
uucp  stream  tcp  nowait  omvskern  /usr/sbin/uucpd
uucpd -l0
```

If you want to have the `uucpd` daemon run with a user ID other than OMVSKERN (for example, UUCPD), you need to decide what the new user ID will be.

Perform the following steps to customize the `uucpd` daemon.

1. Change the line in `/etc/inetd.conf` to:

   ```
   uucp  stream  tcp   nowait  uucpd  /usr/sbin/uucpd uucpd -l0
   ```

   _____

2. Define user ID UUCPD to RACF.

   ```
   ADDUSER UUCPD DFLTGRP(OMVSGRP)
   OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
   NOPASSWORD
   ```

The NOPASSWORD option indicates that this is a protected user ID that cannot be used to enter the system by using a password or password phrase. The user ID will not be revoked due to invalid logon attempts. In this case, you are defining the UUCPD user ID without a TSO/E segment.

_____

When you are done, you have customized the `uucpd` daemon so that it runs with the UUCP user ID.

Customizing the rlogind daemon

The `rlogind` daemon validates rlogin requests. If you choose to have the `rlogind` daemon run with a user ID other than OMVSKERN, you will have to customize it.

Warning: The rlogin daemon is by nature unsecure because it requires user authentication that uses cleartext sockets. A remote attacker can exploit this vulnerability to sniff logins and passwords. Do not use this daemon if possible.

To customize the `rlogind` daemon so that it runs with a user ID other than OMVSKERN (for example, RLOGIND), issue the following:

```
ADDUSER RLOGIND
DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/')
PROGRAM('/bin/sh')) NOPASSWORD
```

The NOPASSWORD option indicates that the user ID is a protected user ID, which cannot be used to enter the system by using a password or password phrase. The user ID will not be revoked due to invalid logon attempts.

Tips:

- For all the other setup steps required for `rlogin`, see “Setting up for rlogin” on page 337.
- If you are writing or porting your own command to process login requests, the shell interface to rlogin is the FOMTLINP module, which is documented in Appendix B, “Modules for the login and logout functions,” on page 411. FOMTLINP has many parameters that can be used to tailor the rlogin processing. FOMTLINP is the login function and FOMTLOUT is the logout function.

For more information, see rlogind in *z/OS UNIX System Services Command Reference*.

Customizing the cron daemon

`cron` is a clock daemon that runs the `at`, `batch` and `crontab` jobs at specified times and dates. `crontab` runs regularly-scheduled jobs, while `at` and `batch` are used for jobs that are run only once.

Before customizing the `cron` daemon, you need to read either “Customizing the cron daemon for the first time” on page 324 or “Migrating from a previous release” on page 325 if you have mounted the root file system read-only.

Customizing the cron daemon for the first time

If you are customizing for the first time, you will have to perform some setup steps. The steps assume that `/usr/spool` and `/usr/lib/cron` do not exist or that they are empty directories. If these assumptions are not true, then you need to follow the instructions in “Migrating from a previous release” on page 325.

1. Set up an `/var/spool` (or `/etc/spool`) directory. Consider using `/var/spool` rather than `/etc/spool` and using a separate file system for the cron spool data.

   Tip: The `/var` directory is the location for IBM products to put their own customization and execution data. It is for IBM product usage only and cannot be edited or modified by you. It is up to you to decide where to put the spool directory. However, because those files are built by `cron` when it runs, they are suitable for the `/var` directory. In addition, because `/etc` typically contains a lot of data and `/var` does not, using `/var` might keep `/etc` from being overloaded.

a. Create the /var/spool directory and set its permissions to 755. For example:

```
mkdir -m 755 /var/spool
chmod 755 /var/spool
```

b. Remove the empty /usr/spool directory, if it exists. For example:

```
rmdir /usr/spool
```

c. Create a symbolic link from /usr/spool to /var/spool. For example:

```
ln -s /var/spool /usr/spool
```

This symbolic link is shipped to you, beginning in z/OS V1R13.

2. Set up a /var/cron directory.

a. Create the /var/cron directory and set its permissions to 755. For example:

```
mkdir -m 755 /var/cron
chmod 755 /var/cron
```

b. Remove the empty /usr/lib/cron directory, if it exists. For example:

```
rmdir /usr/lib/cron
```

c. Create a symbolic link from /usr/lib/cron to /var/cron. For example:

```
ln -s /var/cron /usr/lib/cron
```

3. Create the /var/spool/cron, /var/spool/cron/atjobs and /var/spool/cron/crontabs directories and set their permissions to 755. For example:

```
mkdir -m 755 /var/spool/cron
chmod 755 /var/spool/cron

mkdir -m 755 /var/spool/cron/atjobs
chmod 755 /var/spool/cron/atjobs

mkdir -m 755 /var/spool/cron/crontabs
chmod 755 /var/spool/cron/crontabs
```

When you are done, you have customized the cron daemon for the first time.

Migrating from a previous release

If you are migrating from a previous release, do the following actions:

- Copy existing /usr/spool to /var/spool as described in “Customizing the cron, uucp, and mail utilities for a read-only root file system” on page 123.
- Copy existing /usr/lib/cron to /var/cron as described in “Customizing the cron, uucp, and mail utilities for a read-only root file system” on page 123.

Guideline: If you are moving from a read/write root to a read-only root, follow the steps in “Customizing the cron, uucp, and mail utilities for a read-only root file system” on page 123. If you do not mount the root read-only, the procedure that is described in “Steps for customizing the cron daemon” on page 325 will still work. However, if you are using a shared file system, mounting in read-only mode is suggested.

Steps for customizing the cron daemon

Before you begin, you need to make sure that several files have the correct settings.

1. The **at**, **batch**, and **crontab** executable program files must be owned by a UID(0) user, and the setuid bit must be set. To check the settings, issue:

```
ls -E /bin/at /bin/batch /bin/crontab
```

2. The **cron** executable program file must have the setuid bit off and the program control attribute set. To check the settings, issue:

```
ls -E /usr/sbin/cron
```

Perform the following steps to set up the cron daemon.

1. Copy /samples/queuedefs to /usr/lib/cron/queuedefs.

   For example:

   ```
   cp /samples/queuedefs /usr/lib/cron/queuedefs
   ```

   _____

2. Set up the queue definitions in the /usr/lib/cron/queuedefs file.

   For example:

   ```
   c.5j2n15w
   ```

   c
   Queue name. The default queue name for crontab jobs is**c**.

   5
   Number of jobs running at the same time. Set this to the appropriate value for your installation.

   2
   The nice value mapped to the PRIORITYGOAL or PRIORITYPG statement in BPXPRMxx. Set this to the appropriate value for your installation.

   15
   If five jobs are already running, wait 15 seconds before trying the next job.

   _____

3. Create lists for allowed and denied users for the crontab command:

   - Allowed users: /usr/lib/cron/cron.allow
   - Denied users: /usr/lib/cron/cron.deny

   Tip: To give access to all users, create an empty cron.deny file and do not create an cron.allow file.

   _____

4. Create lists for allowed and denied users for the at command:

   - Allowed users: /usr/lib/cron/at.allow
   - Denied users: /usr/lib/cron/at.deny

   Tip: To give access to all users, create an empty at.deny file and do not create an at.allow file.

   _____

5. Set the TZ environment variable as described in Format of the TZ environment variable in *z/OS UNIX System Services Command Reference*. cron uses this time zone when matching the crontab entries. at jobs use the TZ of the user.

   _____

When you are done, the cron daemon has been customized and can be started.

- It must be started by a user ID with READ permission to the BPX.DAEMON resource profile in the FACILITY class. (For more information about BPX.DAEMON, see “Setting up the UNIX-related FACILITY and SURROGAT class profiles” on page 72 and “Establishing the correct level of security for daemons” on page 313.)

- It is typically called from `/etc/rc`. The command is:

```
_BPX_JOBNAME=CROND /usr/sbin/cron &
```

  As explained in "Using & at the end of a command" on page 329, using the & at the end of a command starts the command in the background and the _BPX_JOBNAME environment variable assigns a job name to the `cron` daemon.
- The `cron` daemon can only be started once because it will continue to run in the background.

For more information about starting daemons, see "Starting daemons" on page 328.

Note:

1. Because `cron` might try to send mail, you might want to configure the `mailx` utility as described in "Customizing electronic mail" on page 217.
2. The job log file is in `/usr/spool/cron/log` and can be used when debugging. The log file must be periodically archived and cleaned up or it will grow too large.

Scheduling at and cron jobs

Schedule cron jobs by using `at` or `crontab`.

1. To run `bigcopy.sh` script at 11:00 p.m., issue:

   ```
   at -f bigcopy.sh 23:00
   ```

   To list the `at` jobs, issue `at -l`.
2. To schedule regular jobs, issue:

   ```
   crontab myjobs
   ```

   `myjobs` contains the following entries:

   ```
   0  0 * * *   /u/admin/daily_bup   >>/etc/bup_log 2>>&1  #midnight daily
   30 1 * * 6   /u/admin/weekly_bup  >>/etc/bup_log 2>>&1  #1:30 every Saturday
   ```

   Input consists of six fields, separated by blanks. All blank lines and any input that contain a # as the first non-blank character is ignored. The first five fields give a date and time in the following form:

   - A minute, expressed as a number from 0 through 59.
   - An hour, expressed as a number from 0 through 23.
   - A day of the month, expressed as a number from 1 through 31.
   - A month of the year, expressed as a number from 1 through 12.
   - A day of the week, expressed as a number from 0 through 6 (with 0 standing for Sunday).

   Any of these fields can contain an asterisk (*) standing for all possible values. For example, if you have an * as the day of the month, the job runs every day of the month.

   The sixth field of a crontab entry is a string that the shell executes at the specified time.

To list the `crontab` jobs, issue `crontab -l`.

Note:

1. `crontab` jobs do not run with the shell variable definitions that exist when `crontab` was invoked. Also, login profiles are not run. The only environment variables that are set are HOME, LOGNAME, PATH, SHELL, and TZ.

   However, `at` jobs inherit the current environment variables.
2. To ensure that user IDs that share a UNIX UID have the correct settings, the user's HOME and LOGNAME environment variables are set according to the MVS identity of the user.

3. PATH is set to the system default. If you want to invoke commands or scripts in other directories, you need to specify the full path name or set the PATH variable in your crontab job.
4. The shell variable is set to `/bin/sh`.

Important: Do not change or put other files in these spool directories:

Starting daemons

Daemons can be started by JCL and also by the shell. Some daemons such as `inetd` can also be started by the shell. Interactive login shells, shell scripts run as background jobs from a login shell, and batch jobs using BPXBATCH to run the shell all can start daemons.

BPXBATSL is provided as an alias for BPXBATCH. BPXBATSL performs a local spawn but does not require the resetting of environment variables. BPXBATSL behaves exactly like BPXBATCH, and allows local spawning whether the current environment is set up or not.

Many daemons can be started from the shell, both interactively and from shell scripts. In general, processes started from the shell complete (either successfully or with some error) before the parent shell itself exits. Any processes still running receive a SIGHUP signal when the parent shell exits. The default action for SIGHUP is to terminate the process. That is, when the shell exits, the system terminates all running processes started by the shell.

Daemon processes are long-running and generally must continue to run even after the invoking shell terminates. Those daemons started using the shell are therefore written to ignore SIGHUP signals. They are also typically written to return control to the shell immediately. If they did not return, the shell script would wait forever for the daemon to exit.

The rules are as follows:

- When started from the shell, most daemons should not be placed in the background environment. That is, an ampersand should not appear on the shell command line that starts a daemon. Doing so exposes the background job containing the daemon to SIGHUP and causes the daemon to terminate unexpectedly when the shell script exits.
- Some daemons either do not protect themselves from the SIGHUP signal or do not return to the shell immediately. You have to have those daemons start in the background environment. To do this, add an ampersand character at the end of the command line that starts the daemon.
- When starting daemons in the background environment, it is very important to include a `sleep` command at the end of the script. This command gives the background processes time to get started and set up to ignore SIGHUP so that when the shell exits, the daemons keep running when the shell script completes. The amount of time required can be determined empirically. A value of 5 seconds is suggested for a start.

A shell script that starts a more simple daemon called `slowpoke` that does not return control immediately to the shell would look like this:

```
slowpoke &
sleep 5
exit
```

In summary, a shell script that starts the `syslogd` and `cron` daemons would look like the following:

```
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf &
_BPX_JOBNAME='CROND' /usr/sbin/cron &
sleep 5
exit
```

Although `cron` and `syslogd` return immediately and protect themselves from SIGHUP, the & is included with `syslogd` because this is the only method of getting _BPX_JOBNAME to take effect.

Using & at the end of a command

Using an & at the end of a command starts the command in the background. The shell forks a child process, executes the command program, and then does not wait for the command to complete. Some daemons must be started this way in order to allow the invoking shell script (such as `/etc/rc`) to continue. **cron** does not need to be started with an & because it forks itself to create the child process, which continues running while the **cron** parent process returns to the invoker such as `/etc/rc`. If the script does `/usr/sbin/cron`, the shell will spawn the **cron** program to create a child process, and then the**cron** program will fork a child process to run the daemon independently. The **cron** command returns to the shell, and the script continues.

However, system programmers might want the **cron** daemon process to have a job name. To do this from a shell script, you can use the _BPX_JOBNAME environment variable. (This can be done on the command line, or in a prior export command.) The _BPX_JOBNAME variable assigns the job name to executed programs, running in forked processes, but not to locally spawned processes. As a result, the shell command

```
_BPX_JOBNAME=CROND /usr/sbin/cron
```

cannot assign the job name to the **cron** daemon. (It depends if the spawn is done within the same address space.) But, the shell command

```
_BPX_JOBNAME=CROND /usr/sbin/cron &
```

will assign the job name to the **cron** daemon, because it is run with a fork/exec.

Starting and restarting daemons

There are several ways to start and restart daemons. The method used depends on the level of control the installation has chosen for daemons.

During initialization

Put the command in `/etc/rc` to start the daemon automatically during initialization. For information about starting programs from `/etc/rc`, see “Customizing /etc/rc” on page 204.

When UNIX systems are initialized (IPLed or restarted), the `/etc/rc` shell script is run to perform system initialization functions and to start daemons. If a daemon terminates, a superuser must restart the daemon.

Tip: You can use `/etc/inittab` instead of `/etc/rc` to start daemons.

The following explanation uses the `syslogd` daemon (which supplies logging functions for programs) as an example of a daemon. Similar steps are required for other daemons.

`syslogd` is typically started from `/etc/rc`.

Example

The _BPX_JOBNAME environment variable is set to assign a job name of SYSLOGD to the `syslogd` daemon. The operator will then have better control over managing the `syslogd` daemon.

```
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf &
```

When `/etc/rc` is started by `/etc/init` or `/usr/sbin/init`, stdin is `/dev/null` and both stdout and stderr are open to the `/etc/log` file. The `/etc/rc` script then invokes the requisite daemons with these files, as such. If the `syslogd` process fails, you could re-IPL, but this would be very disruptive to the users.

Using a cataloged procedure

You can start `syslogd` with a cataloged procedure. For example:

```
//SYSLOGD  PROC
//SYSLOGD  EXEC PGM=SYSLOGD,REGION=30M,TIME=NOLIMIT
//       PARM='POSIX(ON) ALL31(ON)/ -f /etc/syslogd.conf'
//SYSPRINT    DD SYSOUT=*
//SYSIN       DD DUMMY
//SYSOUT      DD SYSOUT=*
//SYSERR      DD SYSOUT=*
//CEEDUMP     DD SYSOUT=*
```

For this `syslogd` cataloged procedure to get control with superuser and daemon authority, you must add an entry to the started procedures table, or define it in the STARTED class.

It is suggested that you assign user ID OMVSKERN to SYSLOGD in the RACF started procedures table. For example:

```
DC    CL8'SYSLOGD'  PROCEDURE NAME
DC    CL8'OMVSKERN' USER ID (to be used for SYSLOGD proc)
DC    CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC    XL1'00'       NOT TRUSTED
DC    XL7'00'       RESERVED
```

For more information about the started procedure table, see The started procedures table (ICHRIN03) in *z/OS Security Server RACF System Programmer's Guide*.

To start `syslogd`, issue the following command from the console:

```
S SYSLOGD
```

Whenever the `syslogd` daemon is deactivated, you can issue this command to restart it.

Using BPXBATCH

You can use a cataloged procedure using BPXBATCH to invoke a daemon program located in the file system.

```
//SYSLOGD PROC
//SYSLOGD EXEC PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//     PARM='PGM /usr/lpp/tcpip/sbin/syslogd -f /etc/syslogd.conf'
//* STDIN and STDOUT are both defaulted to /dev/null
//STDERR DD PATH='/etc/log',PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//     PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

`syslogd` requires superuser and daemon authority.

The JCL in the SYSLOGD PROC invokes BPXBATCH, which sets up the standard file descriptors and environment variables before running `/usr/lpp/tcpip/sbin/syslogd`.

In order to reference the `syslogd` messages from the message catalog files in the file system, you must create a symbolic link to the `syslogd.cat` file. With a superuser ID, in one of the z/OS UNIX shells, issue the following command:

```
ln -s /usr/lpp/tcpip/lib/nls/msg/C/syslogd.cat
 /usr/lib/nls/msg/C/syslogd.cat
```

Setting up security procedures for daemons

Consider setting up security for daemons if you plan to take advantage of z/OS UNIX.

Steps for setting up security procedures for daemons

Before you begin: You need to assume the following:

- You want the added system integrity of having BPX.DAEMON defined.

- Daemons will share the OMVSKERN user ID and be started from `/etc/rc`.

Review the procedure for security as described in “Preparing RACF” on page 46 and “Steps for preparing RACF ” on page 46.

Perform the following steps to define and start daemons.

1. Define the group OMVSGRP.

```
ADDGROUP
OMVSGRP OMVS(GID(1))
```

   _____

2. Define the user OMVSKERN.

```
ADDUSER OMVSKERN DFLTGRP(OMVSGRP)
OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
NOPASSWORD
```

   NOPASSWORD indicates that OMVSKERN is a protected user ID; it cannot be used to enter the system by using a password or password phrase. The user ID will not be revoked due to invalid logon attempts.

   _____

3. Add the daemon cataloged procedure to the RACF STARTED class or the Started Procedure table, module ICHRIN03. Do not make it trusted. See “Steps for preparing RACF ” on page 46.

   _____

4. Create the BPX.DAEMON FACILITY class profile.

```
RDEFINE FACILITY
BPX.DAEMON UACC(NONE)
```

   _____

5. Grant daemon authority to the kernel. Be sure that you know which ID was already permitted to BPX.DAEMON, as part of the security setup described in “Steps for preparing RACF ” on page 46. In this example, the ID is OMVSKERN.

```
PERMIT BPX.DAEMON
CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
```

   _____

6. Activate program control if you have not already done so and ensure that the daemon programs and Language Environment runtime library are in a library that is controlled by z/OS.

```
SETROPTS WHEN(PROGRAM)
RDEFINE PROGRAM * ADDMEM
('CEE.SCEERUN'/RTLPAK/NOPADCHK
'SYS1.LINKLIB'/'******'/NOPADCHK) UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

   Change RTLPAK to the pack that the PDS resides on.

   Tip: You can use `PROGRAM PROFILE **` instead of `PROGRAM PROFILE *`.

   _____

When you are done, you have set up and defined daemons.

Giving daemon authority to vendor-written programs

If you are writing a program that uses z/OS UNIX services to change user identity (such as using setuid and seteuid and so forth), you should refer to “Setting up the UNIX-related FACILITY and SURROGAT

class profiles" on page 72 to determine whether your program's use of these services requires it to have DAEMON authority.

If you determine that your program requires DAEMON authority, then you need to do the following:

1. Document the requirement to assign a user ID to the daemon which has a UID of 0.
2. Document the requirement to permit the daemon to the BPX.DAEMON FACILITY class profile.
3. Document how to start the daemon from `/etc/rc` or as a started procedure.
4. The main program and all programs that it loads must be marked program controlled in the file system or be loaded from an MVS program controlled library. For more information about marking programs program-controlled, see "Customizing the system for IBM-supplied daemons" on page 315. For more information about placing your programs into MVS libraries, see "Moving z/OS UNIX executables into the LPA" on page 359. This section describes steps to move a program into LPA; similar steps can be followed to move a program into a system linklist library or a step library. If you decide to use MVS libraries, you need to also see "Steps for defining programs from load libraries to program control" on page 316.

Tracking down problems when setting up daemons and servers

This section describes possible problems you might encounter when setting up daemons and servers.

This table lists the problems that you might encounter when setting up daemons and servers.

Problem	Reference
A user or daemon is not properly defined with an OMVS segment.	"Verifying the user OMVS segment" on page 332
A group is not properly defined with an OMVS segment.	"Verifying the group OMVS segment" on page 333
A module that is not defined to program control was loaded into the daemon's address space.	"Using external links to access MVS load libraries" on page 334
The daemon module is coming from the file system. The sticky bit might be off or the program might not be in the MVS search order.	"Verifying that the sticky bit is on" on page 333 and "Using external links to access MVS load libraries" on page 334
The executable file being loaded from the file system does not have the program control extended attribute set.	"Finding modules that were not defined to program control" on page 335
The file system containing a program control executable file was mounted with the NOSETUID option. This makes the entire file system uncontrolled.	"Finding modules that were not defined to program control" on page 335
The daemon does not have READ or higher permission to BPX.DAEMON.FACILITY.	"Checking the daemon authority" on page 336
The server does not have READ or higher permission to BPX.SERVER.FACILITY.	"Checking the server setup" on page 336
The in-storage data that is managed by RACF might be out of date.	"Refreshing RACF in-storage data" on page 337
The server might not have been defined to the BPX.SRV.uuuuuuuu SURROGAT class profile.	"Checking the SURROGAT class profile" on page 337

Verifying the user OMVS segment

If a user or daemon is not properly defined with an OMVS segment, you need to verify that the user or daemon has an OMVS segment with a UID.

To verify that the DAEMONU daemon was properly defined with an OMVS segment, issue:

```
LU DAEMONU OMVS
```

You will see output similar to the following one:

```
LU DAEMONU OMVS

  USER=DAEMONU  NAME=UNKNOWN  OWNER=WELLIE    CREATED=92.104
   DEFAULT-GROUP=DAEMONG   PASSDATE=92.125  PASS-INTERVAL=N/A
   ATTRIBUTES=SPECIAL OPERATIONS
    ...
 GROUP=DAEMONG  AUTH=USE CONNECT-OWNER=WELLIE CONNECT-DATE=92.104
      CONNECTS=    82  UACC=NONE     LAST-CONNECT=95.261/14:09:38
  ...
  OMVS INFORMATION
  ---------------
  UID= 0000000000
  HOME= /
  PROGRAM= /bin/sh
  CPUTIMEMAX=NONE
  ASSIZEMAX=NONE
  PROCUSERMAX=NONE
  THREADSMAX=NONE
  MMAPAREAMAX=NONE
```

You should now see that the UID is 0. (The UID for all daemons must be 0, which gives superuser authority to the daemon.)

Verifying the group OMVS segment

If a group is not properly defined with an OMVS segment, you need to verify that the user has a group OMVS segment with a GID defined.

To verify the group OMVS segment of user DAEMONG, issue:

```
LG DAEMONG OMVS
```

You will see output similar to the following:

```
  LG DAEMONG OMVS
   INFORMATION FOR GROUP DAEMONG
       SUPERIOR GROUP=NONE         OWNER=IBMUSER
       ...
       USER(S)=      ACCESS=      ACCESS COUNT=      UNIVERSAL ACCESS=
         DAEMONU       JOIN          000392               READ
         ...

  OMVS INFORMATION
  ---------------
  GID= 0000000500
```

In the output, the UID is 500. (Installations can choose the GID values for their groups.)

Verifying that the sticky bit is on

If the daemon module is in an MVS load library, the file that contains the daemon module must have the sticky bit set on. For information about how to verify that the sticky bit is on, see Table 38 on page 334.

Table 38. Verifying that the sticky bit is on

If you are using	Then
The ISPF shell	From the ISPF shell, enter the file name of the daemon (`/usr/sbin/daemon1`, for example) and request Attributes. You will see a display similar to the following one: `Display File Attributes` `Pathname : /usr/sbin/daemon1` `Link count . . . . : 2` `Set UID bit . . . : 0` `Set GID bit . . . : 0` `Sticky bit . . . . : 1` `...` In the line for the sticky bit, 1 indicates that the sticky bit is on.
The z/OS UNIX shell	Issue `ls -l` You will see a display similar to the following: `-rwxr--r-T 2 SUPERU SYS2 131072` `Oct 25 10:19 daemon1` T indicates that the sticky bit for daemon1 is on.

Note:

1. If the daemon module is in the file system, the file that contains the daemon module must have the program control extended attribute set.
2. If the program does have the extended attribute set, you still need to verify that the file system that it resides in is not mounted with the NOSETUID option. Choose one of the following options:
   - Check the MOUNT statement in BPXPRMxx.
   - Display the file system information by using the **df** command. The file system, the mount table, and ISHELL have attributes that you can use to see this setting:

     `Ignore SETUID . . . . : 1`

     If the "Ignore SETUID" value is set to 1, loading modules from this file system marks your address space dirty. For more information about dirty address spaces, see “Handling dirty address spaces” on page 319.

Using external links to access MVS load libraries

Instead of using the sticky bit for programs that are invoked via either exec() or spawn(), or are loaded with the dllload() function, use an external link to an MVS program name. Both functions search the MVS load library search order for the MVS program.

If you use an external link, the MVS program defined by the external link does not have to be part of the file name of the program that was invoked via either exec() or spawn().

For example, define a z/OS UNIX file `/usr/lpp/internet/bin/wwwss.so` as MVS program name IMWYWWS in an external link.

```
ln -e IMWYWWS /usr/lpp/internet/bin/wwwss.so
```

When you are done, you have created an external link that can be used to access an MVS load library.

Finding modules that were not defined to program control

If a module that was not defined to program control is loaded into an address space and a process in the address space tries to invoke a restricted z/OS UNIX service such as setuid(), you get the JRENVDIRTY reason code. It indicates a dirty address space; see “Handling dirty address spaces” on page 319 for more information.

Steps for finding modules that were not defined to program control

Before you begin, you need to check your job log and have the security administrator check the security console for diagnostic messages.

Perform the following steps to find the module that was not defined to program control.

1. Search the RACF database for a list of the modules that are defined to program control.

   For example, issue the following TSO/E command:

   ```
   SEARCH CLASS(PROGRAM) NOMASK
   ```

   You will see output similar to the following:

   ```
   CEEOLVD
   CEEOV
   CEEPLPKA
   CEEZ24
   DAEMON
   EDCUCSNM
   EDCUEYI1
   EDC$EUEY
   ...
   ```

   ```

   ```

   _____

2. Look for the daemon module (for example, DAEMON) and locations in the format EDC$xxyy (in the output in Step 1, EDC$EUEY is the module for the U.S. English locale).

   _____

3. If the output of the SEARCH module shows *, issue:

   ```
   RLIST PROGRAM *
   ```

   The * covers any module name in the libraries displayed in the output of the RLIST command. If a VOLSER is displayed with a library name, make sure that the VOLSER is also correct.

   _____

4. Gather data about which programs need to be defined to program control by using SLIP. The complete details are in Obtaining traces for program control and Program Access to Data Set (PADS) errors in *z/OS Security Server RACF Diagnosis Guide*.

   Example:

   ```
   SLIP SET,IF,ACTION=TRACE,LPAMOD=(ICHRFR00,xxxxx),J=jobname,
   TRDATA=(STD,REGS,zzzzzz),ML=100,END
   ```

   _____

5. Because this SLIP produces GTF records, you must start GTF. Be sure that you specify PARM TRACE=SLIP. Then use IPCS to format the data with the GTFTRACE IPCS command. You will see output similar to the following:

```
SLIP S+U          ASCB.... 00FAF580 CPU..... 0001     JOBN.... INETD8
                  .....
                  GENERAL PURPOSE REGISTER VALUES
                  0-3..... 7FFEB744 7FFEB748 00000000 007F2978
                  4-7..... 0000000C 007F0738 00000004 007F24D8
                  8-11.... 00000000 7FFEB6A8 80E2323E 007F2978
                  12-15... 00000000 7FFEB6A8 80E23616 0000000C
                  ...

SLIP USR          CPU..... 0001     EXT..... 0001     CNTLN... 00
            0008  C3C5C5C2 C9D5C9E3                    | CEEBINIT
            002C  C3C5C54B E2C3C5C5 D9E4D540 40404040 | CEE.SCEERUN
                  40404040 40404040 40404040 40404040 |
                  40404040 40404040 40404040          |
            0006  D6D7F2D9 E2F1                       | OP2RS1
```

_____

6. Look for a SLIP S+U entry where R15 has a value of 0000000C. Then look at that entry to identify the module and library that needs to be defined to program control.

   _____

You know you are done when you have identified the module and library that needs to be defined to program control.

To define the module to program control, issue:

```
RDEFINE PROGRAM CEEBINIT ADDMEM -
('CEE.SCEERUN'/OP2RS1/NOPADCHK) UACC(READ)
```

Checking the daemon authority

You might need to check to see if the daemon has READ or higher permission to BPX.DAEMON.

To check whether BPX.DAEMON has been properly defined with READ or higher permission, issue:

```
RLIST FACILITY BPX.DAEMON AUTHUSER
```

You will see output similar to the following:

```
RLIST FACILITY BPX.DAEMON AUTHUSER

CLASS      NAME
-----      ----
FACILITY   BPX.DAEMON
 ...
USER      ACCESS   ACCESS COUNT
----      ------   ------ -----
 ...
DAEMONU   UPDATE      000007
 ...
```

The output shows that daemon (DAEMONU in this example) has UPDATE permission to BPX.DAEMON.

Checking the server setup

You can have similar problems setting up a server as when setting up daemons. All of the steps for verifying program control apply to servers as well as daemons, but instead of checking the BPX.DAEMON FACILITY class profiles, verify that the BPX.SERVER profile is properly defined.

To check whether BPX.SERVER has been properly defined with READ or higher permission, issue:

```
RLIST FACILITY BPX.SERVER AUTHUSER
```

You will see output similar to the following:

```
RLIST FACILITY BPX.SERVER AUTHUSER

CLASS      NAME
-----      ----
FACILITY   BPX.SERVER
 ...
USER      ACCESS   ACCESS COUNT
----      ------   ------ -----
 ...
SERVERU   UPDATE      000007
 ...
```

The output shows that BPX.SERVER has been properly defined with READ or higher permission.

Refreshing RACF in-storage data

RACF uses the RACLIST option of the SETROPTS command to define what profile information is to be buffered in storage for faster performance. If you think that you have defined everything correctly, try refreshing the various profiles relating to the daemon and server support.

Following is the set of all relevant refresh commands:

```
SETROPTS WHEN(PROGRAM) REFRESH
SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS RACLIST(SURROGAT) REFRESH
```

Checking the SURROGAT class profile

If your server processes user requests without a password or password phrase, it must be defined to a SURROGAT class profile for the user ID. To check whether the server has been defined to the BPX.SRV.*uuuuuuuu* SURROGAT class profile, use the appropriate RLIST command.

Assuming that your server needs to process requests from user ID ANONYMOS, issue:

```
RLIST SURROGAT BPX.SRV.ANONYMOS AUTHUSER
```

You will get output similar to the following:

```
RLIST SURROGAT BPX.SRV.ANONYMOS AUTHUSER

CLASS      NAME
-----      ----
SURROGAT   BPX.SRV.ANONYMOS
 ...
USER      ACCESS   ACCESS COUNT
----      ------   ------ -----
 ...
SERVERU   READ        000007
 ...
```

From the output, you can verify that the user ID that you are running your server on (in this example, it is SERVERU) has READ or later permission to the BPX.SRV.userid SURROGAT class profile.

Setting up for rlogin

You can use **`rlogin`** to log on to a z/OS UNIX system from a remote system. Two daemons are used when processing rlogin requests:

- The `inetd` daemon handles rlogin requests.
- The `rlogind` daemon is the server that validates the remote login request and checks the password or password phrase. It does not have a customization file in the file system.

The z/OS UNIX system does not use the `.rhosts` file that many UNIX systems use. It indicates the remote hosts and users who can access your system without specifying a password or password phrase. Either a password or password phrase is always required to rlogin to a z/OS UNIX system.

Steps for setting up for rlogin

Before you begin:

1. You need to set up the appropriate security as described in:
   a. “Preparing RACF” on page 46.
   b. “Defining z/OS UNIX users to RACF” on page 51.
   c. “Defining group identifiers (GIDs)” on page 57
   d. “Controlling access to files and directories” on page 83.
   e. “Setting up TCP/IP security” on page 102
2. You also need to customize the SUBFILESYSTYPE statement in BPXPRMxx to include TCP/IP.

Perform the following steps to set up for rlogin.

1. Set up security for the **inetd** and **rlogind** daemons. See “Steps for preparing the security program for daemons” on page 314.

   ---

2. Establish the connection between TCP/IP and z/OS UNIX; see z/OS Communications Server for more information.
   a. Define port 513 in `/etc/services` (if this file is to be used) or in the `hlq.ETC.SERVICES` data set, where *hlq* is the prefix defined by DATASETPREFIX in the TCP/IP profile ("TCPIP" by default). If `/etc/services` is defined, it will be used instead of `hlq.ETC.SERVICES`. The format of the line is:

      ```
      login        513/tcp
      ```

   b. IPL or re-IPL if needed.
   c. Start TCP/IP.

   ---

3. Customize `/etc/inetd.conf`. It tells `inetd` how to handle Internet service requests on Internet sockets. If you do not have that file, copy it from `/samples`.

   For example:

   ```
   cp /samples/inetd.conf /etc/inetd.conf
   ```

   Make these changes in the file:
   a. Change the user ID of the login server (which is `rlogind`) to an ID with daemon authority.

      ```
      login stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
      ```

   b. Comment out any servers that are not needed by putting a # in the first column.

   ---

4. Start the `inetd` daemon. It is typically started from `/etc/rc`, which is executed when the system is initialized. Put these lines in `/etc/rc`, or uncomment them out, as the case might be:

   ```
   _BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf&
   ```

   - When you start `inetd` from the shell, you need to do it from an OMVS session.
   - If TCP/IP is not yet up, you will receive error messages, but `inetd` will try again in a few minutes.
   - To obtain debugging information, issue:

     ```
     /usr/sbin/inetd -d
     ```

- To verify that `inetd` is listening on port 513, issue the TSO NETSTAT INTERVAL command and check the output.

When you are done, you can issue the **`rlogin`** command to log in from a remote UNIX system.

Solving problems with rlogin setup

When debugging problems with rlogin setup, you can use any of the following:

- The `-d` option of the `inetd` command
- The `-d` option of the `rlogind` command
- The `/tmp/.stderr` file

If there are problems on the client side, you might get the following message:

- `Invalid logon name or password`. This message is misleading and often is caused by setup problems on the z/OS UNIX side. It is possible that security was not set up correctly.

If there are problems on the server side, you might get the following messages:

- `Resource temporarily unavailable`. In this case, `inetd` will try initializing the service every three minutes.
- `Service unavailable`. This typically means that the port assignment is not correct. Use the TSO NETSTAT INTERVAL command to verify that OMVS issued a LISTEN for port 513. If port 513 is not there, then `inetd` cannot find the port assignment for 513 in `/etc/services` or *hlq*.ETC.SERVICES. You will need to establish the connection between TCP/IP and z/OS UNIX. Do so by defining port 513 in `/etc/services` (if this file is to be used) or in the *hlq*`.ETC.SERVICES` data set, where *hlq* is the prefix that is defined by DATASETPREFIX in the TCP/IP profile (`"TCPIP"` by default). If `/etc/services` is defined, it is used instead of *hlq*`.ETC.SERVICES`.

Setting up for syslogd to receive messages from the su command

You can configure the **`syslogd`** to receive messages from the **su** command.

Steps for setting up syslogd to receive messages from the su command

Before you begin: You need to have superuser authority in order to start the **`syslogd`** daemon.

Complete these steps to set up **`syslogd`** to receive messages from the **su** command.

1. Set up the **`syslogd`** configuration file `/etc/syslog.conf`.
   a. Create the `/tmp/syslogd` directory.

   ```
   mkdir /tmp/syslogd
   ```

   b. To start logging the **su** command, add one of the following configuration statements to the `syslogd.config` file.

   ```
   auth.*   /tmp/syslogd/auth
   ```

   ```
   auth.notice   /tmp/syslogd/auth
   ```

   If you just want to record the authentication failure of the **su** command, add the following configuration statement.

   ```
   auth.err /tmp/syslogd/auth
   ```

   c. Create the log file.

   ```
   touch  /tmp/syslogd/auth
   ```

2. Start **syslogd**.

```
/usr/sbin/syslogd -f /etc/syslog.conf &
```

   To force **syslogd** to reread the configuration files and activate any modified parameters without stopping, issue the following command.

```
kill -s HUP $(cat /etc/syslog.pid)
```

Now the **syslogd** daemon will receive messages from the **su** command.

For more information about configuring **syslogd**, see Configuring the syslogd daemon in *z/OS Communications Server: IP Configuration Guide*.

Chapter 18. Preparing security for servers

You will read about security for your server applications. The word "server" is taken to mean "server application" , which is an application that provides a service for clients. This server could be part of a software product that will run on any company's z/OS computing environment, or it might be written by your application programmers for your own company's use. This topic is for both audiences:

- The application programmers designing the server. They must decide what kind of security the server is to have so they can code for it and provide documentation (either verbally or in writing) for those who will run the server.
- The security administrator at the company that runs the server. They must set up the profiles based on the documentation provided with the server.

Security administrators, who might not be versed in developing programs, will learn the rationale for setting up profiles in certain ways, and application programmers writing the servers will be able to document the security requirements of their products.

Appropriate decisions need to be made regarding server security. In the past, applications had to run APF-authorized in order to be able to call RACF to build task-level security. z/OS UNIX provides services for servers written in C to create task-level security without being APF-authorized. A server can create a thread-level security environment and control which servers have the ability to do so. You can prepare a z/OS system for a server that uses thread-level security for its clients. (Note that a thread on UNIX systems corresponds to a task on MVS; so, thread-level security is the same as task-level security.)

List of subtasks

Subtasks	Associated procedure
Setting up servers	“Steps for setting up servers” on page 345
Defining servers to process users without passwords	“Steps for defining servers to process users without passwords or password phrases” on page 347

Designing security for servers

Chapter 17, “Setting up for daemons,” on page 311 is prerequisite reading. If you need a high level of security, read and follow the steps in that topic first.

This section is intended for the application developer who is designing and developing servers that use z/OS UNIX. The section describes:

- Setting up the clients with the appropriate security; see “Setting up threads and security” on page 341.
- Controlling access to resources; see “Checking authority to use protected resources” on page 343.
- Using the RACF client ACEE support; see “Limitations of RACF client ACEE support” on page 343.
- Writing the documentation that supports your server; see “Documenting the security requirements” on page 343.

Setting up threads and security

z/OS UNIX supports two fundamental types of application servers: multithreaded servers and single-threaded servers.

- A multithreaded server has multiple sequential flows of control. In this family of applications, the server can process more than one unit of work at a time.

- A single-threaded server has one sequential flow of control. In this family of applications, the server processes one unit of work at a time.

z/OS UNIX provides the pthread_security_np() callable service and support through the C runtime library. It enables unauthorized multithreaded servers to create and delete a RACF security environment in a way that is mediated and controlled by the kernel and RACF. Multithreaded servers can customize the security environment of a thread, thus allowing it to be executed under a different RACF identity than that of the server. You must authorize the server to use that service.

The term *unauthorized* refers to applications that are not APF-authorized and do not run in supervisor state or in a system storage protection key.

A server that uses the pthread_security_np() service can customize the RACF identity of a thread. This server initiates a thread that processes the client's request. If the server customizes the thread that is initiated for the client with the client's RACF identity, any resource access decisions to RACF-protected resources are made by using the client's RACF identity and authorizations.

Depending on the trust you place in a server, you have the option of enforcing whether to use both the server's RACF identity and the RACF identity of the client in resource access control decisions on z/OS.

You can choose one of the following options:

- Only the RACF user ID of the client is used in local resource access control decisions that are made by RACF on z/OS.
- Both the RACF user ID of the server and the RACF user ID of the client are used in local resource access control decisions on z/OS.

The use of the pthread_security_np() service is partly protected by the RACF FACILITY class profile BPX.SERVER. If this profile is defined, then the RACF user ID that is associated with the server needs at least READ authority to use the pthread_security_np() service.

- If the RACF user ID that is associated with a server is permitted with UPDATE access to this profile, the server is allowed to establish a thread-level (task-level) security environment for clients connecting to the server. With UPDATE authority to BPX.SERVER in the RACF FACILITY class, the server can act as a *surrogate* of the client. This means that the identity of the thread that is associated with the request from the server's client executes with the z/OS user ID of the server's client.

  The RACF identity of the client determines the type of access that is allowed to z/OS resources (such as data sets) and z/OS UNIX resources (such as UNIX files), which are accessed by the client's thread in the server.
- READ access allows the server to establish a thread-level security environment for the clients it services. However, the user ID of the server and the user ID of the client must be authorized to the resources the server accesses. A thread-level security environment in which both the client's and server's identities are used in the access control decision, but a password or password phrase was not supplied by the client, is called an *unauthenticated client* security environment.

  Depending on the design and implementation of the client/server application, a client might need to supply an authenticator to the server.

  For example, the client might be prompted to supply a password, password phrase, or a password substitute, such as a RACF PassTicket, to the server to prove its identity. If a RACF password, password phrase, or PassTicket is specified as an option on the pthread_security_np() service, and it is valid for the client user ID, only the RACF user ID of the client is used in rendering access control decisions. This task level security environment that is created by a server is called an *authenticated client* security environment. Because the client has trusted the server sufficiently to supply a RACF password, password phrase, or PassTicket to the server, the server can act as a surrogate for that client.

  This capability enables you to determine on behalf of which user IDs the server can act and what resources the server can access when acting on behalf of one of its clients.

Potentially, for additional security checking, two audit records can be produced to audit the client accessing the resource and the server accessing the resource on behalf of the client.

If you choose to implement this additional security checking, you might need to authorize the identity that is associated with the server to the resource profiles that protect the resources that are accessed by the server on behalf of its clients.

Checking authority to use protected resources

Application developers might want a server to check the authority of a user to access RACF-protected resources. In this way, the server can control access to those resources. The resources include printers and tapes, but not UNIX files and directories and MVS data sets. Use the auth_check_resource_np (BPX1ACK) callable service that z/OS UNIX provides, or the XL C/C++ runtime library check_resource_auth_np() function call to invoke the RACF v_dceauth callable services to do the necessary checking.

Restriction:

- The resources must be defined to RACF general resource classes.
- The server must have read access to the BPX.SERVER FACILITY class profile or have UID(0).
- All server modules must be defined to RACF.

Limitations of RACF client ACEE support

If both the server's RACF identity and the client's RACF identity are used to make access decisions, you should be aware of limitations of the RACF client ACEE support.

- RACROUTE REQUEST=FASTAUTH processing does not check both the server and client RACF identities automatically.

  Unauthorized servers cannot use the RACROUTE REQUEST=LIST instruction to build in-storage profiles for RACF defined resources. Profiles must reside in storage before RACROUTE REQUEST=FASTAUTH can verify a user's access to a resource.
- The client/server relationship is not propagated from the server.

If your server controls access to resources by checking and authenticating both the server's RACF identity and client's RACF identity, treat servers you do not trust as end points on z/OS. These servers should not be allowed to submit batch jobs or use the services of other servers that run exclusively under the identity of the client. You must ensure that servers that do not meet this criteria are not authorized to the profile BPX.SERVER in the RACF FACILITY class.

Documenting the security requirements

In documentation that accompanies your servers, you might need to give some instructions to the security administrator whose installation will be running the server. This might happen if your server uses services that require special authority; these services include:

- The SAF R_dceruid() callable service
- The z/OS UNIX convert_id_np callable service
- The C library function __convert_id_np() function call

Without the appropriate authority set up at the installation, your server will not run. Documentation that accompanies these services tells the security administrator the kind of RACF definitions to set. For example, if the server uses the z/OS UNIX convert_id_np() callable service, the server must have READ access or higher to the IRR.RDCERUID FACILITY class profile.

Establishing the correct level of security for servers

The choice of security level is a decision more likely made by management than by security administrators. That decision depends on answers to the questions "How secure does our company's information need to be?" and "How much do we trust our employees?" Regardless of who makes the decision, it is important that both application developers and security administrators understand the

two levels of security supported by z/OS, and the differences between them. The two levels are: UNIX level and z/OS UNIX level. Read the following descriptions to help you decide which level of security is appropriate for your server.

UNIX level: BPX.SERVER is not defined

If the BPX.SERVER (or BPX.DAEMON) FACILITY class is not defined, your system has UNIX-level security. In this case, the system is less secure. This level of security is for installations where superuser authority has been granted to system programmers. These individuals already have permission to access critical MVS data sets such as PARMLIB, PROCLIB, and LINKLIB. These system programmers have total authority over a system. Server programs that run with superuser authority can issue pthread_security_np() service to change the MVS identity of a thread.

To establish UNIX-level security, assign a UID of 0 to the superuser and assign a UID of 0 to the user ID used for running server programs; for example, DATASRVR.

z/OS UNIX level: BPX.SERVER is defined

There are two z/OS UNIX levels:

- RACF running with enhanced program security, BPX.SERVER defined, and BPX.MAINCHECK defined. You can use BPX.MAINCHECK for any privileged z/OS UNIX application that requires a program controlled environment, because the application uses a privileged z/OS UNIX service that requires one. An example is the __passwd() service, which is used by applications such as telnet and rlogin.
- RACF running with BPX.SERVER defined.

RACF with enhanced program security, BPX.SERVER, and BPX.MAINCHECK

If you enable enhanced program security, and you have any daemons or servers that run execute-controlled programs (MVS programs defined to RACF in the PROGRAM class using EXECUTE authority, or loaded from libraries using EXECUTE authority), then you must define the initial program executed by your daemon or server as a trusted ("MAIN") program to RACF via the PROGRAM class. If this initial program resides in the z/OS UNIX file system, rather than in an MVS library, you will need to move it to an MVS library.

Additionally, you can choose whether to extend the enhanced program security protection to your UNIX daemons and servers that do not make use of RACF execute-controlled programs. Enable this function by defining the profile BPX.MAINCHECK to RACF in the FACILITY class. Again, you need to ensure that the initial program executed by your daemon or server resides in an MVS library and you need to define it to RACF as a PROGRAM with the MAIN attribute.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to the BPX.DAEMON FACILITY class. You will also have to choose the users to whom you give the OMVS security profile segments. To accomplish this, refer to “Steps for preparing the security program for daemons” on page 314.

“Steps for setting up enhanced program security” on page 320 explains how to set up enhanced program security.

RACF with BPX.SERVER or BPX.DAEMON defined

If the BPX.SERVER (or BPX.DAEMON) FACILITY class is defined, your system has z/OS UNIX-level security. In this case, the system is more secure than a traditional UNIX system.

This level of security is for customers with very strict security requirements who need superusers to maintain the file system but who do not want these users to have the authority to change their identities to access existing MVS resources. To accomplish this, take the additional steps described in “Defining servers to use thread-level security” on page 345.

Defining servers to use thread-level security

When the profile BPX.SERVER is defined, there might be two authorization checks:

- The first check authorizes the use of the pthread_security_np() service.
- The second check authorizes for whom the server can establish a security context. This check establishes the scope of users for whom the server can act as a surrogate. See "Defining servers to process users without passwords or password phrases" on page 347 for the steps that are required to enable servers to act as surrogates for their clients when a password or password phrase is not specified on the pthread_security_np() service.

You can also use the BPX.SERVER profile to set the scope of z/OS resources that the server can access when acting as a surrogate for its clients. There are two levels of authority that can be granted to the server using thread-level security services:

- UPDATE access

  Lets the server establish a thread-level (task-level) security environment for clients that are connecting to the server. When the RACF identity of the server has been granted UPDATE authority to BPX.SERVER in the RACF FACILITY class, the server is capable of acting as a surrogate for the client. This means that the identity of the thread that is associated with the request from the server's client runs with the z/OS user ID of the server's client. Access control decisions to z/OS resources (such as data sets) and to z/OS UNIX resources (such as UNIX files) which are accessed by the client's thread in the server are made by using the RACF identity of the client.
- READ access

  Lets the server establish a thread-level security environment for the clients that it services. However, the user ID of the server and the user ID of the client must be authorized to the resources which the server will be accessing. A thread-level security context in which both the client's and server's identity is used in the access control decision and a password or password phrase was not supplied by the client is called an unauthenticated client security context.

  Depending on the design and implementation of the client/server application, a client might have to supply an authenticator to the server. For example, the client might be prompted to supply a password, password phrase, or a password substitute, such as a RACF PassTicket to the server to prove its identity. If a RACF password, password phrase, or PassTicket is specified as a parameter on the pthread_security_np() service, and the password, password phrase, or PassTicket is valid for the client user ID, even if the server's identity has been granted READ access to the profile BPX.SERVER in the RACF FACILITY class, the task level security environment is only used in access control decisions. That is, only the RACF user ID of the client is used in making access control decisions. This task level security environment that is created by a server is called an authenticated client security context. Because the client has trusted the server sufficiently to supply a RACF password, password phrase, or PassTicket to the server, the server is granted the capability of acting as a surrogate for that client (user).

Steps for setting up servers

Before you begin, you will need to know which programs are program-controlled. To identify those programs, use the RACF RDEFINE command, which is discussed in "Customizing the system for IBM-supplied daemons" on page 315.

Perform the following steps each time you add a server.

1. Define all programs that are loaded into an address space that requires server authority, including the server program and any runtime library modules, to program control. For more information about defining programs to program control, see "Defining modules to program control" on page 316.

   _____
2. Assign a user ID to the server and define it to RACF.

For example, assume that the user ID of the server is DATASRVR. Define user ID DATASRVR to RACF.

```
ADDUSER DATASRVR DFLTGRP(OMVSGRP) OMVS(UID(7) HOME('/')
PROGRAM('/bin/sh')) NOPASSWORD
```

You can use the NOPASSWORD option with the ADDUSER command for DATASRVR. This indicates that it is a protected user ID that cannot be used to enter the system by means of a password or password phrase. The user ID will not be revoked due to invalid logon attempts. In this case, you are defining the DATASRVR user ID without a TSO/E segment.

_____

3. Create a cataloged procedure. For example:

```
//DATASRVR PROC
//DATASRVR EXEC PGM=DATASRVR,REGION=0M,TIME=NOLIMIT,
//      PARM='POSIX(ON) ALL31(ON)/ serverparms'
//SYSPRINT    DD SYSOUT=*
```

_____

4. Enable the DATASRVR cataloged procedure to obtain control with the required user identity. To do so, you must either add it to the RACF STARTED class or add an entry to the started procedures table.

   For example, to add an entry to the started procedures table:

```
DC    CL8'DATASRVR' PROCEDURE NAME
DC    CL8'DATASRVR' USERID (ANY RACF-DEFINED USER ID)
DC    CL8'DATASGRP' GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC    XL1'00'       NOT TRUSTED
DC    XL7'00'       RESERVED
```

_____

5. Create the FACILITY class profile for the server.

```
RDEFINE FACILITY BPX.SERVER UACC(NONE)
SETROPTS RACLIST(FACILITY) REFRESH
```

_____

6. Activate program control for the server, if you have not already done so for daemon support.

```
SETROPTS WHEN(PROGRAM)
```

_____

7. Grant a level of authority to the server using thread-level security services. The BPX.SERVER FACILITY class profile controls the server's access to the pthread_security_np() service. There are two choices when setting the server's authority:

   - UPDATE access allows the server to establish a thread-level (task-level) security environment for clients connecting to the server. Decisions about access control for z/OS resources (such as data sets) and to z/OS UNIX resources (such as UNIX files) that are accessed by the client's thread in the server are made using only the RACF identity of the client.

     For example, to give UPDATE access in the BPX.SERVER FACILITY class profile to user ID DATASRVR:

     ```
     PERMIT BPX.SERVER CLASS(FACILITY) ID(DATASRVR) ACCESS(UPDATE)
     SETROPTS RACLIST(FACILITY) REFRESH
     ```

   - READ access allows the server to establish a thread-level security environment for the clients that it services. However, unless the server has specified a valid RACF password, password phrase, or PassTicket on the pthread_security_np() service invocation, the user ID of the server and the user ID of the client are used in resource access control decisions.

     For example, to give DATASRVR server authority for unauthenticated clients:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(DATASRVR) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

If you are installing a product that uses thread-level security services, check the documentation that is supplied with the product to determine if the server requires READ or UPDATE access to the BPX.SERVER profile.

If you grant READ access to the BPX.SERVER profile in the FACILITY class, and the server does not request a password, password phrase, or PassTicket for its clients, both the server's user ID and the client's user ID are used in decisions about resource access control. Additional security administration will have to be performed to ensure that both the server's user ID and the client's user ID were appropriately authorized to the resources that are accessed by the server.

———————————————————————————————————

When you are done, you have set up the server.

For example, to start DATASRVR, issue the following command from the MVS console:

```
S DATASRVR
```

If the DATASRVR daemon is deactivated, you can also issue this command to restart it.

Defining servers to process users without passwords or password phrases

Depending on the design and implementation of a client/server application, a client might not supply an authenticator to the server. For example, some servers process user requests that come from generic user IDs representing anonymous users, or use a method of authentication other than a user ID and password or password phrase combination.

In this case, in which the RACF password, password phrase, or password substitute (such as the RACF PassTicket) is not specified on the pthread_security_np() service invocation, an additional check is made to ensure that the server is authorized to act as the client. z/OS UNIX uses profiles defined to the RACF SURROGAT class to authorize the server to act as a surrogate of a client. Profiles defined to the SURROGAT class are of the form:

```
BPX.SRV.<userid>
```

<userid> is the MVS user ID of the user that the server will act as a surrogate of. See “Defining servers to use thread-level security” on page 345 for the steps to authorize a server to act as a surrogate for client user IDs.

Some servers have the requirement to process user requests that come from generic user IDs representing anonymous users. In order for servers to process requests for thread-level security without passwords or password phrases, follow the steps shown below.

Steps for defining servers to process users without passwords or password phrases

Before you begin, you need to identify all MVS user IDs that the specified server needs to access without any client authentication. You also need to determine the level access, either ACCESS(READ) or ACCESS(UPDATE) that the server will have while running with the client's identity. “Defining servers to use thread-level security” on page 345 describes those two levels of authority.

Perform the following steps to define servers to process users without passwords or password phrases. The steps are for a sample server called DATASRVR that can support user ID ANONYMOS without a password or password phrase. As you add more servers, you will need to follow similar procedures.

1. Activate the SURROGAT class support in RACF, if it has not already been set up on your system.

   ```
   SETROPTS CLASSACT(SURROGAT)
   ```

   You only have to do this once on your system.

   If a daemon or server you are running will use the SURROGAT support, consider using the RACLIST command to keep the SURROGAT profiles in storage. The following example shows how to cache the SURROGAT profiles in storage:

   ```
   SETROPTS RACLIST(SURROGAT)
   ```

   ———————————————————————————————————————

2. If the SURROGAT profile is in the RACLIST, any changes to the SURROGAT profiles must be followed by a REFRESH command. To create the SURROGAT class profile for user ANONYMOS, issue:

   ```
   RDEFINE SURROGAT BPX.SRV.ANONYMOS UACC(NONE)
   SETROPTS RACLIST(SURROGAT) REFRESH
   ```

   A similar SURROGAT profile is required for each user ID that a server must support without a password or password phrase.

   ———————————————————————————————————————

3. Permit the server to create a thread-level security environment for a specified user.

   For example, to permit server DATASRVR to create a thread-level security environment for user ANONYMOS, issue the PERMIT command:

   ```
   PERMIT BPX.SRV.ANONYMOS CLASS(SURROGAT) ID(DATASRVR) ACCESS(READ)
   SETROPTS RACLIST(SURROGAT) REFRESH
   ```

   ———————————————————————————————————————

When you are done, you have defined a server to process users without passwords or password phrases.

Chapter 19. Monitoring the z/OS UNIX environment

You can monitor performance, use of resources, and the use of system resources by users and programs. Use the information that is collected to tune the system. Tuning the system might improve performance and reduce resource consumption.

Reporting on activities using SMF records

System management facilities (SMF) collects data for accounting. SMF job and job step accounting records identify processes by user, process, group, and session identifiers. Fields in these records also provide information about resources used by the process. SMF file system records describe system events such as file open, file close, and file system mount, unmount, quiesce, and unquiesce.

You can use SMF to report on the following activities:

- User applications
- Jobs and job-step basis
- Mounted file systems and files

See *z/OS MVS System Management Facilities (SMF)* for the full contents of SMF records provided for z/OS UNIX and for information about how to obtain the records.

SMF record type 30

SMF record type 30 reports activity on a job and job step basis. Even though file system activity is included in the EXCP count for the address space, the process section in the record breaks down the EXCP count into the following categories:

- Directory reads
- Reads and writes to regular files
- Reads and writes to pipes
- Reads and writes to character special files
- Reads and writes to network sockets

This section also provides information about file system lookups, which can use significant resources on systems with hierarchical files.

You can monitor the file system activity of various classes of users by postprocessing SMF type 30 records. This type of monitoring might be helpful in forecasting DASD and other system resource requirements. If key jobs appear to be doing many lookups, your installation might be able to reduce this processing overload. To reduce the overload, reorganize the file system so that key files are closer to the root of the file system.

Applications can reduce lookup activity by using the `chdir` command to change the working directory and specifying only the file name when opening a file.

SMF records contain a program name field for job steps that are initiated by fork(), spawn(), or exec(). For interactive commands, this feature allows performance analysts to determine what resources were required to complete a particular command.

If a user runs the OMVS command with the SHAREAS option or sets the environment variable _BPX_SHAREAS to YES, two or more processes might be running in the same address space. In this case, SMF provides process identification only for the first process in the address space. However, resource consumption is accumulated for all processes that are running.

With an exec that follows a setuid(), the exec processing no longer creates a substep. Instead, the initiator stops the old job (ending type 30 record). Then a new job is started with the user ID that was established on the setuid().

The CPU time for each syscall is accumulated for the process and saved in field SMF30OST. The number of requested z/OS UNIX syscalls is reported in field SMF30OSC. Data for SMF30OST and SMF30OSC is only collected when parmlib option SYSCALL_COUNTS is set to YES.

When SYSCALL_COUNTS is set to NO, the CPU time and the count of syscalls is not accumulated. If you always run with SYSCALL_COUNTS=NO, both SMF30OST and SMF30OSC are always reported with a value of zero.

If you switch between SYSCALL_COUNTS=YES and SYSCALL_COUNTS=NO while collecting SMF data for an address space, the SMF 30 record data will not be accurate.

SMF record types 34 and 35

When a new address space is created as a result of a fork() or spawn() service, SMF cuts a Type 34 record. When the process ends, SMF cuts a Type 35 record. Type 34 is defined as TSO Logon and Type 35 is defined as TSO logoff. If you do not have Type 34 or Type 35 active, you do not need to take any further action. If you do use Type 34 and Type 35 for TSO accounting, then you need to suppress these recordings for UNIX processes.

Example

To suppress these records, add the following:

```
SYS(TYPE(34,35))
SUBSYS(OMVS,NOTYPE(34,35))
```

SMF record type 74

SMF record type 74, subtype 3, reports kernel activity. For more information, see *z/OS MVS System Management Facilities (SMF)*.

SMF record type 80

SMF record type 80 includes an extended-length relocate section. For specifics on auditing information in SMF record type 80, see RACROUTE REQUEST=AUDIT in *z/OS Security Server RACROUTE Macro Reference*.

SMF record type 92

SMF record type 92 provides reports of activities that are related to the z/OS UNIX file system.

SMF subtype 2 and subtype 4 records are written for zFS when it is running in one of the following situations:

- zFS is running in a non-shared file system configuration.
- zFS is running in a shared file system configuration and is mounted NORWSHARE.

Tip: The File System I/O counts that are displayed in the ISHELL mount table is available if type 92 subtype 5 (unmount) is active at the time the file system was mounted. To avoid the overhead that is associated with recording type 92 subtype 10, 11, and 14 (open, close, delete, or rename), adjust the parameters in BPXPRMxx. Use the TYPE or NOTYPE operands to exclude the subtype 10, 11, and 14 records.

Table 39 on page 351 lists the subtypes for SMF record type 92.

Table 39. Subtypes for SMF record type 92. This table lists the subtypes for SMF record type 92 and explains how they are produced.

Subtype	When reports are produced
1	Reports are provided after the file system is mounted. I/O activity for a file system is reported in a subtype 5 record when the file system is unmounted. This activity is not accumulated unless subtype 5 is active when the file system is mounted. When a file system is mounted, SMF begins collecting accounting data for the file system's I/O activity if subtype 5 for unmount is active at the time of the mount. Partial SMF accounting does not occur; either all the information for a file system is collected, or none is collected. Subtype 1 records are useful because they provide information about the total space available in the file system and the total space currently used. You can see if it is time to increase the size of a mountable file system.
2	Reports are provided after the file system is quiesced (that is, suspended). zFS file systems are quiesced during the HSM backup in the following situations: • When the zFS file system is in a non-shared file system configuration. • When the zFS file system is in a shared file system configuration and the file system is mounted with the NORWSHARE option. Any non-zFS system can also be quiesced by programs that call the BPX1QSE (quiesce) callable service. Any file system might be quiesced during certain sysplex operations such as when a file system is moved within a shared file system configuration.
4	Reports are provided after the file system is unquiesced (that is, resumed).
5	Reports are provided after the file system is unmounted. Unmount records provide the following I/O data summarized for the entire mountable file system: • Directory reads • Read and write callable services that were requested • Read and write EXCP counts • Total bytes read and bytes written To obtain this data, SMF recording for unmount must be active at the same time that the file system is originally mounted.
6	Reports are provided after the file system is remounted. If a file system is remounted to change modes between read-only and read/write, a subtype 6 record is produced. It contains the same information as in the type 5 (unmount) record. In order for the remount records to contain data on I/O activity, unmount recording must be active at the time the file system is originally mounted.
7	Reports are provided after the file system is moved. If ownership of a file system is changed in a shared file system configuration, a subtype 7 record is produced.

Table 39. Subtypes for SMF record type 92. This table lists the subtypes for SMF record type 92 and explains how they are produced. *(continued)*

Subtype	When reports are produced
8	Reports are provided after the file system is migrated from the source file system to the target file system. During file system migration, when the target file system is mounted, a subtype 8 record is produced. The I/O activity for this file system is reported in a subtype 5 record when the file system is unmounted. This activity is not accumulated unless subtype 5 is active at the time when the target file system is mounted. Partial SMF accounting does not occur; either all the information for a file system is collected, or none is collected.
10	Reports are provided after a file is opened. I/O activity for particular open is reported in a subtype 11 (close) record when the file is closed. This activity is not accumulated unless subtype 11 is active when the file is opened. When a file is opened, SMF begins collecting accounting data for I/O activity that is related to this open if subtype 11 is active. Partial SMF accounting does not occur; either all the information for an open file is collected, or none is collected. Because collecting file activity can be expensive, collect subtype 10 records only when file-level data is needed.
11	Reports are provided when a file is closed. File-close records provide information about I/O activity of a user or application against a specific file. These records provide the following data for a specific user or application and a specific file: • Read and write callable services that were requested. • Read and write EXCP counts. • Total bytes that are read and bytes that are written. • Path name of the file. To obtain a record of I/O activity, SMF recording must be active for subtype 11 at the time a file is both opened and closed. Because collecting file activity can be expensive, collect subtype 11 records when file-level data is needed.
12	Reports are provided after mmap() is used to establish a mapping between a process's address space and a file.
13	Reports are provided after munmap() is used to remove the mapping that was established by a previous mmap() request.
14	Reports are provided after a file or file directory is deleted or renamed. To gather information about the deleting of files and directories, you can set up monitoring for SMF type 92 records with subtype 14. When files or directories are deleted, you will receive information about the time the file or directory was deleted or renamed, in addition to its file type, serial number, and unique device number. In a shared file system, the recording occurs on the user's system where the command was issued.

Table 39. Subtypes for SMF record type 92. This table lists the subtypes for SMF record type 92 and explains how they are produced. *(continued)*

Subtype	When reports are produced
15	Reports are provided when the security attributes for APF-authorized programs, shared library programs, or programs that are defined to program control are changed. External calls to change these attributes are audited. The various places where the system clears these attributes, such as if the file is opened for write or if it is renamed, are not audited. In addition to the regular user, file, and file system information for type 92 records, the following information is also included • The old and the new security attributes, to show what was changed. • The file's RACF file ID, to help in correlating these records with the XXXX_FILE_ID field of SMF 80 records for the same file. • The full path name of the file. If the path name string that was passed on the call to chattr() is an absolute path name, it is copied here. If it is a relative path name, the input string is appended to the realpath() value of the current working directory. The path name will be as passed on the call to chattr() for absolute path names. For relative path names the input path name segment is appended to the realpath() value of the current working directory.
16	Reports about the activity of sockets, character special files, pipes, and FIFOs are provided when files are closed. File-close records provide information about I/O activity of a user or application against a specific file. These records provide the following data for a specific user or application and a specific file: • Read and write callable services that were requested • Read and write EXCP counts • Total bytes that are read and bytes that are written • Path name of the file To obtain a record of I/O activity, SMF recording must be active for subtype 16 at the time a file is both opened and closed. Because collecting file activity can be expensive, only collect subtype 16 records when file-level data is needed.
17	Reports that contain information about the number of times a file is accessed throughout the life of an open are provided. The information is written on the SMF global recording interval and when the internal representation (control block) of the file is freed. In some error flows (for example, file system move failures and dead system recovery), some SMF records might be lost.

Unmount records also provide the following I/O data summarized for the entire mountable file system:

- Directory reads
- Read and write callable services that were requested
- Read and write EXCP counts
- Total bytes read and bytes written

File-close records provide information about I/O activity of a user or application against a specific file. These records provide the following data for a specific user or application and a specific file:

- Read and write callable services that were requested
- Read and write EXCP counts
- Total bytes that are read and bytes that are written

- Path name of the file

Monitoring process activity

To monitor process activity, you can display pending jobs. You can also enable applications to monitor activities of z/OS UNIX processes.

While z/OS UNIX is being started or restarted, jobs are not processed. After the start or restart process has been completed, jobs will begin processing again. An operator message tells the system operator that jobs are waiting for z/OS UNIX to become available again.

The system operator can also use the D OMVS,A=DUBW command to show which jobs are in wait status. When the BPXO040I message is displayed in response, the PID field shows a "-" instead of a PID value and the STATE field is set to D, indicating that the job is waiting to be dubbed. For example:

```
*10.52.00 STC00010 *BPXP022E ONE OR MORE JOBS ARE WAITING FOR UNIX SYSTEM
 * SERVICES AVAILABILITY.

   - 10.52.10     d omvs,a=dubw
 00  10.52.12       BPXO040I 10.52.10 DISPLAY OMVS 266           C
     OMVS     000E SHUTDOWN
     USER     JOBNAME  ASID    PID    PPID STATE   START     CT_SECS
              TC       0022     -        0  1D----              .001
```

Using installation exits

You can use installation exits to enable applications to monitor process activities.

Preprocess initiation exit (BPX_PREPROC_INIT)
Receives control immediately before the creation of any new process. This exit cannot use any z/OS UNIX callable service. When these exit routines receive control, the Process Exit Data Block (PEDB) will contain data about the creating process.

Post-process initiation exit (BPX_POSPROC_INIT)
Receives control immediately after the creation of any new process. When this exit receives control, the Process Exit Data Block (PEDB) will contain the creator and the new process data.

Process image initiation exit (BPX_IMAGE_INIT)
Receives control immediately before any new process image is initiated. This occurs after a successful spawn, attach_exec, attach_execmvs, exec, or execmvs callable service is done. The exit will receive control before the new process image file is run. When this exit receives control, the Process Exit Data Block (PEDB) will contain the data of the creator and the new image.

Preprocess termination exit (BPX_PREPROC_TERM)
Receives control immediately before the termination of a process. These exits might receive control in the address space of the process or in the master address space, if the address space of the process was terminated. In the latter case (ASID=1), the exit cannot use z/OS UNIX callable services. When these exits receive control, the Process Exit Data Block (PEDB) will contain details about the terminating process.

Exit routines can be added to each exit point. z/OS UNIX passes control to the exit routine when an exit point is reached. Information about the current process and its creator is then passed to the exit routine.

Defining exits

The kernel defines the four process start and end exits at kernel initialization time by means of the CSVDYNEX service.

When you are adding exit routines to an exit, certain exit attributes are required.

For BPX_PREPROC_INIT, BPX_POSPROC_INIT, and BPX_IMAGE_INT:

- AMODE=31

- REENTRANT=REQ
- PERSIST=IPL
- ABENDNUM=10000
- ABENDSCONSEC=YES
- FASTPATH=YES,KEY=0

For BPX_PREPROC_TERM:

- AMODE=31
- REENTRANT=REQ
- PERSIST=IPL
- ABENDNUM=10000
- ABENDSCONSEC=YES
- FASTPATH=NO,KEY=0

Adding exit routines to exits

You can use any of the following methods to add exit routines to exits:

- PROGxx member of SYS1.PARMLIB
- SETPROG console command
- REQUEST=ADD via the CSVDYNEX service

Example 1

To add the DUBEXIT exit routine to the BPX_PREPROC_INIT exit by using a PROGxx member:

```
EXIT ADD
     EXITNAME(BPX_PREPROC_INIT)
     MODNAME(DUBEXIT)
```

Example 2

To remove the DUBEXIT exit routine from the BPX_PREPROC_INIT exit:

```
SETPROG EXIT,DELETE,EXITNAME=BPX_PREPROC_INIT,MODNAME=DUBEXIT,FORCE=YES
```

Chapter 20. Tuning performance

You need to take some tuning steps because you are combining MVS and UNIX. Two tuning situations exist, depending on how your system is being used: as a production system or a porting system. For both, you can take important steps and control resource consumption.

Tip: If your system is running in a virtual server or as a VM guest, the storage size should be at least 64 MB.

List of subtasks

Subtasks	Associated procedure
Caching UID and GID information in VLF	"Steps for caching UID and GID information in VLF" on page 358
Moving an executable in the file system into the LPA	"Steps for moving an executable in the file system into the LPA" on page 359
Setting process limits	"Steps for setting process limits in z/OS UNIX" on page 368
Changing process limits	"Steps for changing the process limits for an active process" on page 371

Improving performance of runtime routines

When C programs (including the shell and utilities) are run, they frequently use routines from the Language Environment runtime library, which come from the SCEERUN data set. On average, about 4 MB of the runtime library are loaded into memory for every address space running a Language Environment-enabled program, and copied on every fork. If you have 200 address spaces running, this uses 800 MB of pageable storage. It also increases your paging rates or reduces the amount of work that the system can support.

To reduce this overhead and improve performance, you can take one or more of the following actions, beginning with a configuration that includes CEE.SCEERUN and CEE.SCEERUN2 in the LNKLST concatenation:

- Put the SCEELPA data set in the LPA list. Because the SCEERUN data set has many modules that are not reentrant, you cannot place the entire data set in the link pack area by using the LPALIST member of SYS1.PARMLIB. However, you can take advantage of a SCEELPA data set that contains a subset of the SCEERUN modules that are reentrant, reside above the line, and are heavily used by z/OS® UNIX.

  Use the LPALSTxx member to place the SCEELPA data set in the LPA list. You can also add more modules to the LPA list by using the dynamic LPA capability (SET PROG=). For more information about those members, see:

  - LPALSTxx (LPA library list) in *z/OS MVS Initialization and Tuning Reference*
  - LNKLSTxx (LNKLST concatenation) in *z/OS MVS Initialization and Tuning Reference*
- Put libraries CEE.SCEERUN and CEE.SCEERUN2 into LLA. The library lookaside facility (LLA) minimizes I/O by keeping heavily used modules in a virtual lookaside facility (VLF) data space and keeping a version of the library directory in its own address space. You might choose to include all LNKLST modules in LLA. For example:

```
SYS1.PARMLIB(CSVLLAxx)
    LIBRARIES(-LNKLST-)
    FREEZE(-LNKLST-)
```

Tuning tips for the compiler utilities

On systems where application development is a critical activity, you can improve the performance of the compiler utilities c89, cc, cxx,c++ by loading CBC.SCCNCMP into dynamic LPA if space allows. If common area space is an issue, selected modules can be put into the LPA.

If space allows, load the entire load library into dynamic LPA.

```
SYS1.PARMLIB(PROGxx)
        .  .  .
        LPA      ADD MASK(*)    DSNAME(CBC.SCCNCMP)
         .  .  .
        LNKLST ADD NAME(LLZB) DSNAME(CEE.SCEERUN)
        LNKLST ADD NAME(LLZB) DSNAME(CEE.SCEERUN2)
        .  .  .
```

For more information about the PROGxx parmlib member, see PROGxx (authorized program list, exits, LNKLST sets and LPA) in *z/OS MVS Initialization and Tuning Reference*.

Improving performance by updating the PROGxx member

On systems where application development is the primary activity, making certain changes to the PROGxx member of SYS1.PARMLIB might improve performance. For example:

```
SYS1.PARMLIB(PROGxx)
 LPA,ADD,MODNAME(CEEBINIT,CEEBLIBM,CEEEV003,EDCZV),DSNAME(CEE.SCEERUN)
 LPA,ADD,MODNAME(IEFIB600,IEFXB603),DSNAME(SYS1.LINKLIB)
```

Your primary Language Environment level must be the default level for the release, and you must be using the default compiler. To verify your Language Environment primary level, check that the library name (for example, CEE.SCEERUN) appears first in the linklist concatenation in the LNKLSTxx member of SYS1.PARMLIB.

Caching RACF user and group information in VLF

Caching UIDs and GIDs improves performance for commands such as `ls -l`, which must convert UID numbers to user IDs and GID numbers to RACF group names. RACF allows you to cache UID and GID information in virtual lookaside facility (VLF).

To avoid performance degradation, VLF should be made active. For more information about performance considerations, see “RACF performance considerations” on page 50.

Steps for caching UID and GID information in VLF

Before you begin, you need to have access to the COFVLxx member of SYS1.PARMLIB.

Perform the following steps to cache UID and GID information in VLF.

1. Add these VLF options to the COFVLFxx member of SYS1.PARMLIB.

```
CLASS NAME(IRRUMAP)
  EMAJ(UMAP)
CLASS NAME(IRRGMAP)
  EMAJ(GMAP)
CLASS NAME(IRRSMAP)
  EMAJ(SMAP)
```

   _____

2. Start VLF, specifying the updated member.

In this example, the updated member is COFVLF33.

```
START VLF,SUB=MSTR,NN=33
```

```
_____________________________________________________________________
```

When you are done, you have cached GID and UID information in VLF.

Because VLF is started after RACF and OMVS, you might get a message from RACF during the IPL saying that running without VLF will cause slower performance. If VLF is being started, you can ignore this message.

For information about updating the VLF parmlib member COFVLFxx, see “COFVLFxx ” on page 35.

Moving z/OS UNIX executables into the LPA

Some executables in the file system can be commonly used by many concurrent users, or they can be loaded and deleted frequently during normal production. Such executables are performance sensitive, and they might be good candidates for inclusion in the LPA. Moving such programs to the LPA can reduce storage consumption, reduce DASD I/O activity for loads, and reduce the storage copied on each fork().

One thing to consider when you analyze which executables belong in LPA is that modules with the sticky bit on are not eligible for local spawn(). If your executable is normally invoked by spawn(), either by the shell or by another application, turning on the sticky bit forces spawn() processing to execute the program in a spawned child address space. In cases where local spawn() would be used if the sticky bit were not on, this reduces the benefit of loading the executable from the LPA.

Steps for moving an executable in the file system into the LPA

Before you begin, you need to know how long the executable or DLL name is.

Perform the following steps to move an executable in the file system into the LPA.

1. Select one of the following actions, depending on how long the executable or DLL name is.

If . . .	Then . . .
The executable or DLL name is no more than 8 characters excluding the extension (such as longname.dll) and contains no special characters that are not valid for TSO PDSE member names.	a. Bind the executable or DLL into a PDSE member (for example, LONGNAME) b. For the executable or DLL in the file system, turn on the sticky bit. For example: `chmod +t longname.dll` c. Verify that the executable is marked reentrant. You can check that the executable is marked reentrant by checking TSO browse on the PDSE, locating the member in the member list, pressing PF11 and then looking for the RN attribute. d. Put the executable into dynamic LPA by modifying the PROGxx parmlib member or by issuing the SETPROG console command.

If . . .	Then . . .
The executable or DLL name is more than 8 characters long, excluding the extension (for example, reallylonglongname.dll), or if the name contains special characters.	a. Bind the executable or DLL into a PDSE member with a valid member name (for example, REALLY) b. Rename the original executable or dll to save it. For example: `mv reallylonglongname.dll reallylonglongname.dll.save` c. Create an external link for the name. For example: `ln -e REALLY reallylonglongname.dll` d. Verify that the executable is marked reentrant. Check that the executable is marked reentrant by checking TSO browse on the PDSE, locating the member in the member list, pressing PF11 and then looking for the RN attribute. e. Put the executable into dynamic LPA by modifying the PROGxx parmlib member or by issuing the SETPROG console command.

When you are done, you have moved an executable in the file system into the LPA.

Binding the executable or DLL into a PDSE

To bind the executable or DLL into a PDSE you can use the following sample JCL. While this JCL will work for most simple executables, the binder options (specified as PARM= below) will not be appropriate for all executables or DLLs. If you have a makefile for the executable or DLL, this will tell you what binder options should be used.

Because most executables in the file system today are program objects (new load module format), they must be bound into PDSE libraries. So, SYSLMOD DD should point to a PDSE (Data Set Name Type = Library).

```
//PUTINLPA  JOB  MSGLEVEL=(1,1)
//*                                                            *
//* INLMOD DD STATEMENT SPECIFIES THE DIRECTORY THAT CONTAINS  *
//* THE PROGRAM.                                               *
//*                                                            *
//* THE INCLUDE STATEMENT SPECIFIES THE NAME OF THE FILE TO    *
//* RUN FROM THE LPA.                                          *
//*                                                            *
//* THE NAME STATEMENT SPECIFIES THE FILE NAME BUT IN          *
//* UPPERCASE. THIS MUST BE SAME AS THE FILE NAME.             *
//*                                                            *
//LINK     EXEC PGM=IEWL,REGION=100M,
//  PARM='LIST,XREF,LET,RENT,REUS,AMODE=31,RMODE=ANY,CASE=MIXED'
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//INLMOD   DD PATH='/bin/'
//SYSLMOD  DD DSN=OECMD.LPALIB,DISP=SHR
//SYSLIN   DD *
         INCLUDE INLMOD(myprog)
         ENTRY   CEESTART
         NAME    MYPROG(R)
/*
```

Use an SMP/E USERMOD to link any IBM-supplied programs from a UNIX file system into another library, such as when loading it into LPA. Doing so automatically keeps the two copies of the module at the same level when service is installed. It also provides a record of modifications to your systems. For more information about SMP/E usermods, see *z/OS SMP/E User's Guide*.

Also, not all modules are eligible for LPA. Modules placed in LPA must be both reentrant and executable. For more information about PROGxx, see PROGxx (authorized program list, exits, LNKLST sets and LPA) in *z/OS MVS Initialization and Tuning Reference*.

Using the shared library extended attribute

Shared object libraries contain subroutines that can be shared by multiple processes. Programs using shared libraries contain references to the library routines that are resolved by the loader at run time. For information about shared object library programs and the ST_SHARELIB extended attribute, see loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Executables that have the ST_SHARELIB extended attribute turned on are called *system shared library programs*. They are an optimal way of sharing large executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single-page table (similar to LPA). The storage used in the user address space to establish the mapping to the shared library region is from the high end of private storage; in most cases, it does not interfere with the virtual storage used by the application program.

The amount of storage that is carved out of the high end of private storage of each address space that loads a system shared library object is based on the value of the SHRLIBRGNSIZE parameter in the BPXPRMxx parmlib member. If this value is set too high, the storage set aside for the mapping of the shared library region might interfere with the private storage requirements of each of these address spaces. For this reason, the value specified for SHRLIBRGNSIZE should be the minimum size that is required to contain all of the shared library programs that are to be used on the system. Note that z/OS UNIX attempts to map the entire SHRLIBRGNSIZE into the private region, not just the portion that contains programs. If the private region is too small to map the entire SHRLIBRGNSIZE, then this shared library region is not be used, A message is not issued to indicate that the SHRLIBRGNSIZE was not mapped.

See “Defining UNIX files as shared library programs” on page 319 for information about setting the ST_SHARELIB extended attribute.

Tuning tips for the file system

Use the following tips to fine-tune your file system:

- Make sure that all files in your file system have valid owning UIDs and GIDs. If you restore files from an archive and accidentally keep a UID and GID that were valid on another system, it can create problems that affect response time. For example, say that there is an invalid UID associated with a file. When you use a utility that checks the UID (such as `ls -l`), RACF searches the entire database for the UID.
- Place zFS data sets on volumes that are cached with DASD Fast Write.
- Give each user a separate mountable file system. Doing so enables you to avoid I/O contention by spreading user file systems across multiple DASD devices.
- Use the temporary file system (TFS) for `/tmp`.
- Customize the statements in the BXPRMxx member of SYS1.PARMLIB as needed. See “Customizing the BPXPRMxx member of SYS1.PARMLIB” on page 17 for guidelines and tips.

Tuning limits in BPXPRMxx

You can improve the performance of your z/OS UNIX environment by fine-tuning your BPXPRMxx member. However, because each installation is unique, some of the suggestions might not be appropriate for your system.

For more information, refer to these documents:

- *z/OS MVS Planning: Workload Management*
- *z/OS MVS Initialization and Tuning Guide*

- *z/OS MVS Initialization and Tuning Reference* for parmlib members
- *z/OS RMF User's Guide* for RMF monitoring
- *z/OS RMF Report Analysis* for RMF reports

Monitoring system and process limits

You can monitor the status of the z/OS UNIX system and process limits with the D OMVS, LIMITS operator command, and console messages that indicate when limits are reaching critical levels. Use SET OMVS or SETOMVS to change certain system limits dynamically, or SETOMVS with PID= to change a process-level limit for a specific process.

The LIMMSG statement in BPXPRMxx controls message activity for limits checking. You can specify whether no console messages are to be displayed when any of the parmlib limits have been reached (NONE). That is, console messages are to be displayed for all processes that reach system limits and for certain process limits (SYSTEM), or they are to be displayed for all the system limits and the process limits (ALL).

The LIMMSG options can be changed with the SETOMVS LIMMSG command. The LIMMSG value appears in the D OMVS,O display.

If the LIMMSG statement is specified with SYSTEM or ALL, a warning console message appears whenever a limit reaches 85%, 90%, 95%, and 100%, identifying the process that has reached the limit. When the limit reaches the next limit level, the prior message is removed from the console and a new message indicates the new limit level that was reached. When the limit falls under the 85% threshold, a message indicates that the resource shortage was relieved.

Changing from LIMMSG(ALL) or LIMMSG(SYSTEM) to LIMMSG(NONE) with the SETOMVS command stops any further monitoring of resources. However, existing outstanding messages are not deleted from the screen for a process until the limit is relieved for that process.

Tip: When LIMMSG(ALL) is in effect, a large number of messages can be issued. This option is best suited for use during the initial configuration of a system, when the installation has not yet determined the optimal settings for the z/OS UNIX parmlib limits.

Monitoring use of system resources

z/OS UNIX provides the system programmer with a number of controls that monitor and tune the use of system resources by users. As part of the tuning process, you can follow these initial rules of thumb:

1. Assume that each user will consume up to double the system resources required for a TSO/E user.
2. Assume that at most 4 PTYs will be required per average user.
3. Assume that the starting point for maximum processes per user is 25.
4. Assume that 4 concurrent processes will be required by the average active user.
5. Assume that 5 processes will be required for various daemons.
6. Assume that 3 concurrent address spaces will be required by the average active user. This number will be high if your users are running with the _BPX_SHAREAS environment variable set to YES.
7. Assume that no user, including servers or daemons, needs more than 2048 files open at one time for Unicode Services conversion. In this case, the default MAXIOBFUSER setting of 2048 (which equals 2 G of above the bar storage) is enough and no changes to the initial setting is necessary.

Tip: If you have a few users who need a large number of processes, use the PROCUSERMAX keyword in the OMVS segment to set the process limit for these users.

For example, assume that your system supports 600 TSO/E users and has enough capacity for 20 additional users. Rather than adding more TSO/E work, you want to allow TSO/E users to access z/OS UNIX. You have no other z/OS UNIX work on your system at this time. In this example the initial settings in BPXPRMxx might be:

```
MAXUIDS(20)
MAXPTYS(80)
```

```
MAXPROCUSER(25)
MAXPROCSYS(85)
```

Table 40 on page 363 shows how the initial settings were calculated.

Table 40. Calculating initial settings when tuning process activity. This table lists the initial settings for certain BPXPRMxx parameters.

Parameter	Initial setting	Note
MAXUIDS	20	If you allow 20 current TSO/E users to access the z/OS UNIX system, each of them could consume twice the resource they normally used for TSO/E. This would require all your remaining system resources.
MAXPTYS	80	Assume that 4 PTYs are needed per user. Users can login with multiple sessions at the same time
MAXPROCUSER	25	This should normally be a reasonable starting point. Some users might require more processes, depending on the work they are doing. This value can be set only on a system-wide basis.
MAXPROCSYS	85	Assume that you need 4 processes per user and 5 processes for daemons. (20 users * 4) + 5 daemons = 85 processes.

Controlling dispatching priorities

The setpriority() and chpriority() functions let the caller set the dispatching priority for a process, a process group, or a user. The priority value that is specified can range from -20 to 19. On this scale, -20 is highest priority and 19 is lowest priority. The nice() function allows a calling process to change its own priority.

Resulting nice() values can range from 0 to 39, with 0 being the highest priority and 39 being the lowest. With all three services, appropriate privileges are required to increase the priority of one or more processes.

Priority values (-20 to 19) and nice() values (0 to 39) are mapped one-to-one such that nice() values are always 20 higher than priority values. All processes start with a priority value of 0 and a nice() value of 20.

```
     priority value                    nice value
(setpriority and chpriority)              (nice)
----------------------------            ----------
  -20    A                                   0
    .    | higher priority                   .
    .    |                                   .
    .    |                                   .
    0    -- start here                      20
    .    |                                   .
    .    |                                   .
    .    | lower priority                    .
  +19    V                                  39
```

In general, do not enable nice(), setpriority(), and chpriority() support. Instead, you should only use normal SRM controls. However, nice(), setpriority(), and chpriority() support is provided. This support interfaces with SRM and workload management to provide system control and monitoring support.

If your installation plans to support the **cron** daemon, setpriority() support might be needed. **cron** allows interactive users to schedule work to run in the background at various times in the future. Normally, this background work should run at a lower priority than other interactive work. By default, **cron** uses setpriority() to lower the priority of batch work it starts. The return code is not checked, so if the setpriority() call fails, the batch work runs at the same priority as other forked children. This could become a problem if background work started by **cron** begins to affect the responsiveness of foreground

interactive work. In this case, it might be appropriate to customize your system to support three levels of dispatching priority (as illustrated in the following example).

To enable the nice(), setpriority(), and chpriority() functions, an installation must specify a PRIORITYPG statement or PRIORITYGOAL statement in BPXPRMxx. The first value corresponds to a priority value of -20 (very high priority). The next corresponds to a priority value of -19, and so on until the 40th value corresponds with a priority value of 19. If fewer than 40 values are specified, the last value is propagated through the remaining priority values. The same performance group can be specified several times.

Installations that are running in goal mode to exploit MVS workload management (WLM) can enable nice(), setpriority(), and chpriority() support using the PRIORITYGOAL statement in the BPXPRMxx parmlib member. They must specify a service class for each possible priority value (-20 to 19). If fewer than 40 service classes are specified, the last service class is propagated to all remaining priority values. The same service class can be specified several times. All service classes that are specified must appear in your current service policy.

Tip: Do not specify PRIORITYPG and PRIORITYGOAL in BPXPRMxx unless you need nice() and setpriority() support. It is simplest and best to give MVS full control over priorities of work.

System limits and process limits

Limits can be set for the system (system-wide limits) or for individual processes (process limits).

System-wide limits, which are limits that apply to every process, are set in the BPXPRMxx member of SYS1.PARMLIB. You can display limits defined in BPXPRMxx by using the operator commands D OMVS,OPTIONS and D OMVS,LIMITS.

- Limits associated with an MVS application derive their value from MVS. The soft and hard limit might be different when an MVS unit of work is dubbed.
- Limits for processes initiated by z/OS UNIX that cause an identity change, such as telnet, rlogin or a daemon process using setuid or exec, are created with the same hard and soft limits.

Process limits are limits that apply to individual processes. You can set some process limits by using the RACF user profile segment. Some process limits, such as the disk space, allow for a file or the size of a dump are set in BPXPRMxx.

- You can control the amount of resources used by specified z/OS UNIX users by setting individual limits for these users, as described in “Setting limits for users” on page 58. These limits apply to all users except for those with a UID of 0. Normally, when a process is initially dubbed, the soft limit is inherited from MVS and the hard limit is set in the BPXPRMxx parmlib member.

  For more information about UID(0) superuser authority, see “Obtaining security information about users” on page 56. Users with UID(0) will still have a limit but they can change it while other users can only change their soft limits.
- You can set limits on a process for resources such virtual storage space after you know which resources the application will need, and how the operating system affects application limits. For information about the types of processes and how they are created, see “Setting process limits in z/OS UNIX” on page 367.
- You can use the RACF ADDUSER and ALTUSER commands to specify the ASSIZEMAX limit on a per-user basis. See the topic on Limiting the use of private memory objects in *z/OS MVS Programming: Extended Addressability Guide*, which discusses the use of MEMLIMIT. Table 43 on page 368 lists the process limits that you can set.

Before setting process limits as described in “Setting process limits in z/OS UNIX” on page 367, you need to understand how z/OS UNIX applications interact with the operating system and take into consideration all services that affect that resource. Then you can decide what soft and hard limits the application needs. For an explanation soft and hard limits, see “What are hard limits?” on page 365 and “What are soft limits? ” on page 365.

What are hard limits?

The hard limit is the maximum value that a process's application can raise a soft limit to. The hard limit is derived from z/OS UNIX or RACF.

- Use BPXPRMxx statements to define z/OS UNIX defaults. Defaults exist for z/OS UNIX processes even when none are defined in BPXPRMxx. To find out what the defaults are, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.
- Limits defined in the RACF user profile. See Table 43 on page 368 for a list of hard limits that are defined in the RACF user profile.

z/OS UNIX mechanisms that affect limits are setrlimit(), prlimit(), inheritance from the parent and spawn inheritance structure (BPXYINHE), identity change, and dubbing. Any process can raise the soft limits to the hard limits. A superuser can raise both the hard and soft limits. A fork/spawn child inherits the same limits unless the parent changed the limits in the spawn inheritance structure or there is an identity change. The SETOMVS operator command can also affect these settings.

What are soft limits?

The soft limit is the value of the current process limit that is enforced by the operating system. If a failure such as an abend occurs, the application might want to temporarily change the soft limit for a specific work item, or change the limits of child processes that it creates. For example, a larger server daemon might reduce the amount of virtual memory available to a spawned child. New processes receive the same limits as the parent process as long as the installation or the application do not alter those values and an identity change does not occur. The soft limits for CPUTIME, ASSIZE and MEMLIMIT can be affected by several MVS limits mechanisms.

MVS limits are the soft limits provided to z/OS UNIX processes when z/OS UNIX services are invoked using TSO login, STC, or JCL. At the first request for a kernel service, the system dubs the program as a z/OS UNIX process. When a traditional MVS unit of work is first dubbed, the soft limits are normally obtained from MVS. The hard limit is normally obtained from BPXPRMxx if it is higher than the soft limit.

New processes that are created by a dubbed user receive the same soft and hard limits as the parent process if the installation has not changed the process limits and an identity change has not occurred.

How are limits handled after an identity change?

When the installation does not use any other method to define a limit, then exec and spawn handle limits after an identity change. After an identity change, an exec sets hard and soft limits to z/OS UNIX values. The change to z/OS UNIX values can also be accomplished by using a setuid() followed by an exec().

Spawn() can also cause an identity change. A spawned child that was created with a different identity than the parent using InheUserid or _BPX_USERID sets the hard and soft limits to the z/OS UNIX values for the new identity. A forked child with no identity change inherits the settings of the parent.

Task-level security (pthread_security or _login) causes spawn to use the limits of the new identity.

Resources values that processes receive when they are dubbed a process use the RACF profile to determine the hard limit if it is higher than the soft (current) limit. It is also used when processes are initiated by a daemon process using an exec after setuid(). In this case, both the RLIMIT_AS hard and soft limits are set to the address-space-size value.

Inheriting soft limits

Normally, soft and hard limits are not changed when a new process is created unless the creating process requests a change. Both the soft limits and the hard limits are set to the value that was specified the parent's inheritance structure, if it is requested. The values of both limits are raised to the hard limit if a child was spawned with a new identity, or if an exec occurred after the identity was changed.

Setting a global value in BPXPRMxx (using the MAXASSIZE parameter) provides the values that the system will use to assign to all processes. The RACF security administrator can override these values by defining the MEMLIMIT value for individual identities in the associated OMVS segment.

The only time MEMLIMIT or ASSIZE from the OMVS segment is used to define these limits for inheritance is with a spawn with identity change or an exec after setuid. For more details, see "What happens when an identity change occurs?" on page 366.

What happens when an identity change occurs?

When an identity change occurs, the new identity might or might not have an OMVS segment associated with it. If an identity change takes place, the following happens:

1. If the new identity has the limit defined by RACF, that limit overrides any other system control including the IEFUSI installation exit and the parent's inheritance structure.
2. If there is no RACF segment for the new identity, the IEFUSI installation exit might override the inheritance structure and set the limit.
3. If a RACF limit was not set for the new identity and if IEFUSI did not set the limit, the Inhe inheritance overrides the current values of the parent's inheritance structure.
4. The setting of the MAXASSIZE parameter in BPXPRMxx is the default for a new z/OS UNIX identity if none of the other values applied (except for MEMLIMIT, which is set in SMFPRMxx).

Setuid() alone or child processes created by either fork or spawn after a setuid retains the limits behavior of the previous identity.

What happens if an identity change does not take place when a child is created?

Normally, the child inherits the parent's current hard and soft limits. However, a change to the child's limit causes the child's hard and soft limit to be set to the hard limit. The child's limit can be changed by the parent in the inheritance structure on spawn() or by IEFUSI. How the system honors changes made in the inheritance structure or IEFUSI will vary depending on the parent's security setting.

When the same limit is not defined in the OMVS segment, the system will honor IEFUSI even if the parent requests a change using the inheritance structure (Inhe).

After a limit has been defined for a parent identity by the security administrator, that limit is trusted before other system components only if it is higher than the MVS limits. When a limit is defined in the OMVS segment:

- The system ignores requests by IEFUSI to change the limit in the OMVS segment if the limit in the OMVS segment is higher than the limit set in IEFUSI. The OMVS segment is in control of ASSIZE and MEMLIMIT. If the OMVS segment value is lower than IEFUSI, then IEFUSI sets hard and soft limits for both MEMLIMIT and ASSIZE. IEFUSI is in control of ASSIZE and MEMLIMIT.
- If the OMVS segment is in control of ASSIZE and MEMLIMIT for the parent, the system also ignores requests by IEFUSI to change limits for child processes that were created by that identity. In this case, the child is created with the parent's current limits. If IEFUSI is in control of ASSIZE and MEMLIMIT for the parent, the child's MEMLIMIT and ASSIZE will be set by IEFUSI. In both cases, if IEFUSI is active in the system and is attempting to control limits for an OMVS address space, the inheritance structure (Inhe) is ignored for MEMLIMIT and ASSIZE.

What happens if an identity change does not take place when a new process image is created by exec()?

When a new process image is created, limits are not affected unless a limit has been defined in the OMVS segment, an IEFUSI exit modifies the limit, or an identity change occurs. When the limit is controlled by system security or IEFUSI, the hard and soft limit are changed to the hard limit in the new image.

Note: If RACF or UNIX limits for CPUTIMEMAX, ASSIZEMAX, or MEMLIMIT are higher than the MVS limits, they are used for the hard limit.

Specifying a new identity

You can specify a new identity in the following ways:

- Inheritance structure
- The _BPX_USERID environment variable
- The pthread_security callable service
- _login
- The setuid() function

Setting process limits in z/OS UNIX

System-wide limits are defined in the BPXPRMxx parmlib member. Table 41 on page 367 lists the BPXPRMxx statements that you can use to set system-wide limits. If you specify the SHRLIBMAXPAGES parameter, it is accepted but will not have any impact on the system. The value that you specify will never be reached, because user-shared library objects are no longer supported.

Table 41. System-wide limits that can be defined in BPXPRMxx. This table lists the system-wide limits that can be defined in the BPXPRMxx parmlib member.

IPCMSGNIDS	IPCSHMMPAGES	MAXPTYS
IPCMSGQBYTES	IPCSHMNSEGS	MAXRTYS
IPCMSGQMNUM	IPCSHMSPAGES	MAXSHAREPAGES
IPCSEMNIDS	IPCSHMNIDS	MAXUIDS
IPCSEMNSEMS	MAXASSIZE	SHRLIBMAXPAGES
IPCSEMNOPS	MAXMMAPAREA	SHRLIBRGNSIZE

Process-level limits are defined in the BPXPRMxx parmlib member. Table 42 on page 367 lists the BPXPRMxx statements that you can use to set process-level limits.

Table 42. Process-level limits that can be defined in BPXPRMxx

MAXASSIZE	MAXFILEPROC	MAXPROCSYS
MAXCORESIZE	MAXPIPEUSER	MAXQUEUEDSIGS
MAXCPUTIME	MAXPROCUSER	MAXTHREADS

MAXCPUTIME specifies the RLIMIT_CPU hard limit resource values that processes receive when they are dubbed a process. RLIMIT_CPU indicates the CPU time that a process is allowed to use, in seconds. The soft limit is obtained from MVS. If the soft limit value from MVS is greater than the MAXCPUTIME value, the hard limit is set to the soft limit. This value is also used when processes are initiated by a daemon process using an exec after setuid(). In this case, both the RLIMIT_CPU hard and soft limit values are set to the MAXCPUTIME value.

- For processes running in or forked from TSO or BATCH, the MAXCPUTIME value has no effect. A superuser can override this value by specifying a new time limit in the spawn inheritance structure on __spawn().
- For processes running in or forked from TSO or BATCH, the MAXCPUTIME value has no effect. The TIME limit is inherited from the parent. If a TIME parameter is specified on the JCL for the started task, then that value is used. If not, then the TIME value is taken from the JES default TIME value.
- For processes created by rlogind or another daemon, MAXCPUTIME is the time limit for the address space.

Table 43 on page 368 lists the hard limits that can be defined in the RACF user profile.

Table 43. Hard limits that can be defined in the RACF user profile. This table lists the hard limits that can be defined in the RACF user profile.	
Hard limit	**Description**
ASSIZEMAX	The maximum address space size (RLIMIT_AS) for the z/OS UNIX user.
CPUTIMEMAX	The maximum CPU time (RLIMIT_CPU) for the z/OS UNIX user.
FILEPROCMAX	The maximum number of files per process for the z/OS UNIX user.
MEMLIMIT	The maximum storage above the bar (nonshared memory size). MEMLIMIT is initially set in the SMFPRMxx parmlib member.
MMAPAREAMAX	The maximum below the bar memory map size for the z/OS UNIX user.
PROCUSERMAX	The maximum number of processes per UID for the z/OS UNIX user.
SHMEMMAX	The maximum size of shared memory
THREADSMAX	The maximum number of threads per process for the z/OS UNIX user.

Steps for setting process limits in z/OS UNIX

Before you begin, you need to understand how the MVS limits affect z/OS UNIX processes. For an explanation of soft and hard limits, see "What are hard limits?" on page 365 and "What are soft limits? " on page 365. You can modify some limits. Parameters are provided by batch JCL, TSO logon, and started to limit region size and high memory. Some MVS system-wide limits such as MEMLIMIT are initially set in SMFPRMxx.

You also need to have planned the system-wide limits that will affect all z/OS UNIX users. For more information, see "Defining system limits" on page 24.

Perform the following steps to set process limits in z/OS UNIX.

1. Determine the sources that your installation uses to control MVS limits and z/OS UNIX limits. For more details, see "Tuning limits in BPXPRMxx" on page 361. For more information about limiting the use of memory objects, see Limiting the use of private memory objects in *z/OS MVS Programming: Extended Addressability Guide*.

   _____

2. Specify the z/OS UNIX system-wide limits in BPXPRMxx. The system-wide limits are listed in Table 41 on page 367. If you do not set them, defaults are used. To look up the defaults, see BPXPRMxx (z/OS UNIX System Services parameters) in *z/OS MVS Initialization and Tuning Reference*.

   _____

3. Specify higher limits for individual processes, if needed. For an explanation about setting process limits in general, see "Setting limits for users" on page 58. For a discussion of the use of MEMLIMIT, see Limiting the use of private memory objects in *z/OS MVS Programming: Extended Addressability Guide*.

   You can use any of the following to change z/OS UNIX limits:

   - The SETOMVS operator command
   - z/OS UNIX programming APIs
   - RACF
   - z/OS UNIX commands
   - prlimit
   - setrlimit
   - spawn()

You can use the RACF ADDUSER and ALTUSER commands to specify the ASSIZEMAX limit on a per-user basis. Table 43 on page 368 lists the process limits that you can set.

To define or change information in the OMVS segment of a user profile, including your own, you must have the SPECIAL attribute or at least UPDATE authority to the segment through field-level access checking.

For more information about the OMVS segment in RACF user profiles and a complete list of what you can specify, see The OMVS segment in group profiles in *z/OS Security Server RACF Security Administrator's Guide*.

Resource values that processes receive when they are dubbed a process will use the RACF profile to determine the hard limit if it is higher than the soft (current) limit. It is also used when processes are initiated by a daemon process using an exec after setuid(). In this case, both the RLIMT_AS hard and soft limits are set to the address-space-size value

When you are done, you have set process limits in z/OS UNIX.

Note: Limits defined in the RACF user profile or modified by the IEFUSI installation exit override limits defined by z/OS UNIX processes.

Using the IEFUSI installation exit to set process limits

The IEFUSI and IEALIMIT installation exits control region size and memory above the bar, but z/OS UNIX ignores changes made by IEALIMIT. Changes made by IEFUSI to the region limit and hi memory limit are used if a RACF user profile is not used to define these values for the user. The IEFUSI exit can change the values used by the system for virtual storage available above and below the 16M line, and hi memory. Normally, z/OS UNIX takes action to ensure that these changes are honored. When a limit is specifically defined for a user via the OMVS segment, z/OS UNIX will ignore the changes made by the IEFUSI exit if the ASSIZE or MEMLIMIT in the OMVS segment is specified as a higher value than IEFUSI.

Because z/OS UNIX limits are normally inherited from MVS, the IEFUSI exit has already had a chance to modify the region size and MEMLIMIT for TSO, batch, and started task. The exit does not have to be called again during OMVS subsystem processing. z/OS UNIX processes such as telnet and rlogin do not have a chance to change the limits, so IEFUSI is not needed to control those limits.

If you install your own IEFUSI exit, update your SMFPRMxx parmlib member to exclude OMVS work and use z/OS UNIX to control the process limit. Specify:

```
SUBSYS(OMVS,NOEXITS)
```

Started subtasks such as OMVS, BPXOINIT, and colony address spaces fall under SUBSYS STC. These address spaces might be subject to IEFUSI limitations if IEFUSI exits are allowed for SUBSYS STC. IBM strongly recommends that you always set REGION=0 and MEMLIMIT=NOLIMIT for OMVS, BPXOINIT, and colony address spaces.

Message IEE968I is issued when the SET SMF= command is processed because z/OS UNIX does not support the SSI Notify function. You can ignore this message.

After a hard limit is defined in the user RACF profile, the parents' hard and soft values for that limit will override IEFUSI, MVS, and z/OS UNIX changes in any child processes or executed programs. An executed or spawned process after an identity change always set hard and soft limits to the OMVS limit of the new identity. Other processes exhibit this behavior only if the value in the user RACF profile was higher than the value provided by MVS when the process was dubbed.

For more information about the IEFUSI installation exit, see IEFUSI — Step initiation exit in *z/OS MVS Installation Exits*.

Displaying process limits

You can use any of the following to display process limits:

- ps shell command

- `ulimit` shell command
- D OMVS,LIMIT operator command
- D OMVS,OPTIONS
- MEMLIMIT
- ASSIZEMAX

You can display the RACF limit using the following RACF command:

```
LU user NORACF OMVS
```

For more information about how to obtain security information for users, see "Obtaining security information about users" on page 56. You can obtain the information if the security administrator has set up field-level access for users for the OMVS segment of the RACF user profile as described in "Setting up field-level access for the OMVS segment of a user profile" on page 56.

Example

Issue:

```
ps
```

Following is a sample output:

```
/shut/home/wellie2==>ps  ; alias to ps -o ruser,pid,vsz,vsz64,vszlmt64,comm
 RUSER  PID  VSZ VSZ64  VSZLMT64 COMMAND
   Erin  24 2432  0     20M sh -L
   Erin  29 2432  0     20M /tst/bin/ps -oruser,pid,vsz,vsz64,vszlmt64 -oargs
```

Example

Issue:

```
ulimit -a
```

Following is a sample output:

```
core file         8192b
cpu time          unlimited
data size         unlimited
file size         unlimited
stack size        unlimited
file descriptors  256
address space     259192k
memory above bar  17592186040320m
```

Example

Assuming that the PID is 0050331651:

```
d omvs,limits,pid=0050331651
```

Following is a sample output:

```
BPXO051I 16.42.32 DISPLAY OMVS 277
OMVS     000D ACTIVE          OMVS=(6F)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     MEGA1    0020 0050331651   33554434 1CI--- 16.31.46      .051
  LATCHWAITPID=         0 CMD=sh -L
PROCESS LIMITS:        LIMMSG=NONE
                    CURRENT  HIGHWATER    PROCESS
                      USAGE      USAGE      LIMIT

MAXFILEPROC             6          7        256
MAXFILESIZE           ---        ---    NOLIMIT
MAXPROCUSER             3          4    NOLIMIT
MAXQUEUEDSIGS           0          1       1000
MAXTHREADS              0          0        200
MAXTHREAD          0          0       1000
IPCSHMNSEGS             0          0        500
MAXCORESIZE           ---        ---    4194304
MAXMEMLIMIT             0          0     16383P
```

If you do not specify the PID, the display will show the system-wide limits

Changing process limits

You can set a system-wide limit using these BPXPRMxx statements and then specify the process limit using the RACF ADDUSER or ALTUSER command. The OMVS operator command can also be used.

- MAXASSIZE (see “MAXASSIZE” on page 25
- MAXCPUTIME (see “MAXCPUTIME” on page 25)
- MAXFILEPROC (see “MAXFILEPROC” on page 25)
- MAXMMAPAREA (see “MAXMMAPAREA” on page 26)
- MAXPIPEUSER (see “MAXPIPEUSER” on page 26
- MAXPROCUSER (see “MAXPROCUSER” on page 27)
- MAXTHREADS (see “MAXTHREADS” on page 28)

To change the MEMLIMIT of a specific process (or to be more exact, to change the MEMLIMIT value of the ASID that the process is running in), use the SETOMVS command.

Steps for changing the process limits for an active process

Before you begin, you need to know the PID of the process. To find the process, issue D OMVS,A=ALL.

Perform the following steps to change the process limits for an active process.

1. Display information about the current parmlib limits for a process with the ID *nnn*.

   For example, issue:

   ```
   d omvs,limits,pid=nnn
   ```

   You will get a display similar to the following:

```
BPXO051I 16.42.32 DISPLAY OMVS 277
OMVS     000D ACTIVE          OMVS=(6F)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     MEGA1    0020 0050331651   33554434 1CI--- 16.31.46      .051
  LATCHWAITPID=         0 CMD=sh -L
PROCESS LIMITS:        LIMMSG=NONE
                   CURRENT  HIGHWATER    PROCESS
                     USAGE      USAGE      LIMIT

MAXFILEPROC            6          7        256
MAXFILESIZE          ---        ---    NOLIMIT
MAXPROCUSER            3          4    NOLIMIT
MAXQUEUEDSIGS          0          1       1000
MAXTHREADS             0          0        200
MAXTHREADTASKS         0          0       1000
IPCSHMNSEGS            0          0        500
MAXCORESIZE          ---        ---    4194304
MAXMEMLIMIT            0          0     16383P
```

_____

2. Change the high memory limit, MEMLIMIT, in the address space containing PID *nnn*.

   For example, issue:

   ```
   SETOMVS PID=nnn, memlimit=16M
   ```

   You will get a message that the SETOMVS command was successful.

   _____

3. Check that the limit has been changed.

   For example, issue:

   ```
   D OMVS,LIMITS,PID=nnn
   ```

   You will see a display similar to the following:

```
BPXO051I 16.44.40 DISPLAY OMVS 283                                    C
OMVS     000D ACTIVE          OMVS=(6F)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     MEGA1    0020 0050331651   33554434 1CI--- 16.31.46      .051
  LATCHWAITPID=         0 CMD=sh -L
PROCESS LIMITS:        LIMMSG=NONE
                   CURRENT  HIGHWATER    PROCESS
                     USAGE      USAGE      LIMIT

MAXFILEPROC            6          7        256
MAXFILESIZE          ---        ---    NOLIMIT
MAXPROCUSER            3          4    NOLIMIT
MAXQUEUEDSIGS          0          1       1000
MAXTHREADS             0          0        200
MAXTHREADTASKS         0          0       1000
IPCSHMNSEGS            0          0        500
MAXCORESIZE          ---        ---    4194304
MAXMEMLIMIT            0          0        16M *
```

_____

When you are done, you have changed the process limit.

Reference information

For more information, see the following:

- “Defining z/OS UNIX users to RACF” on page 51
- “Storing user-specific information in OMVS segments” on page 53
- “Setting limits for users” on page 58
- “Steps for obtaining security information about users” on page 56

- For information about the RACF ADDUSER command, see ADDUSER (Add user profile) in *z/OS Security Server RACF Command Language Reference*.
- For information about the RACF LISTUSER command, see LISTUSER (List user profile) in *z/OS Security Server RACF Command Language Reference*.
- *z/OS MVS System Management Facilities (SMF)*
- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS Programming: Extended Addressability Guide*

Improving performance of the z/OS shell

You have several options if you want to fine-tune performance of the z/OS shell. You can use the _BPX_SHAREAS and _BPX_SPAWN_SCRIPT . Other helpful hints include controlling the use of STEPLIBs and ensuring that the sticky bit is on.

Do not specify PRIORITYPG and PRIORITYGOAL in BPXPRMxx unless you need nice() and setpriority() support. It is simplest and best to give MVS full control over priorities of work.

Setting _BPX_SHAREAS and _BPX_SPAWN_SCRIPT

To improve the performance of z/OS UNIX shell utilities, use the following environment variables: _BPX_SHAREAS and _BPX_SPAWN_SCRIPT. Note that they cannot be used for the tcsh shell.

- Set _BPX_SHAREAS to YES. (REUSE is the same as YES.) The shell will run foreground processes in the same address space as the shell is running in, which saves the overhead of a fork() and exec().

  To improve performance for all shell users, `/etc/profile` or `$HOME/.profile` should set BPX_SHAREAS=YES as follows:

  ```
  export _BPX_SHAREAS=YES
  ```

  The spawn() runs faster, the child process consumes fewer resources, and the system can support more resources. However, when running multiple processes with BPX_SHAREAS=YES, the processes cannot change identity information. For example, setuid() and setgid() will fail. You cannot execute setuid() or setgid() in the same address space as another process. Also, when the parent ends, the child will end because it is a subtask.

  If the extended attribute for the shared address space is not set, the program will not run in a shared address space, regardless of the setting of _BPX_SHAREAS. The attribute is set by `extattr +s` and reset by `extattr -s`. If the attribute is set, _BPX_SHAREAS has precedence.
- To improve performance when running the shell scripts, set _BPX_SPAWN_SCRIPT to YES. The spawn() service will run files that are not in the correct format to be either an executable or a REXX exec as shell scripts directly from the spawn() function. Because the shell uses spawn() to run foreground commands, setting this variable to YES eliminates the additional overhead of the shell invoking fork after receiving ENOEXEC for an input shell script.

  To provide this performance benefit to all shell users, set the environment variable in `/etc/profile` or `$HOME/.profile`:

  ```
  export _BPX_SPAWN_SCRIPT=YES
  ```

  However, there might be exceptions, depending on your environment.

Tip: Because spawn() uses system resources that require the user's private storage, excessive use might lead to storage shortages in the user's address space.

Controlling use of STEPLIBs

You can improve shell performance by controlling the use of STEPLIBs. A STEPLIB is a set of private libraries used to store a new or test version of an application program, such as a new version of a

runtime library. To improve performance of the z/OS shell, avoid propagating STEPLIBs by using one of the following options:

- If you enter the OMVS command either from ISPF or with STEPLIB data sets allocated, include the statements in the shell profile. For example:

```
if [ -z "$STEPLIB" ] &&; tty -s;
then
        export STEPLIB=none
        exec sh -L
fi
```

  The STEPLIB data sets are not extensively searched or propagated from the shell process to the shell command on exec(). They are also not propagated to shell processes, which might have been necessary because a specific release level of the Language Environment runtime library is needed.

- If you use the OMVS command to login to the shell, you can improve performance by using a logon procedure that does not contain any JOBLIB or STEPLIB DD allocations. Using that procedure reduces the amount of storage that is copied for fork(). It also prevents excessive searching of STEPLIB data sets and the propagation of STEPLIB data sets from the shell process to the shell command processes on exec().

  If you enter the OMVS command from ISPF or with STEPLIB data sets allocated, you can put certain statements in either `/etc/profile` or `$HOME/.profile`.

- If you export a specific STEPLIB, you can have the Language Environment runtime library (SCEERUN) data set allocated as part of ISPLLIB to invoke OMVS from ISPF. In this case, you need to customize `$HOME/.profile` so that subset of the STEPLIB data sets is propagated

  For example, customize `$HOME/.profile` so that only the STEPLIB data set CEE.SCEERUN containing the Language Environment runtime library is propagated.

```
if [ -z "$STEPLIB" ] &&; tty -s;
then
        export STEPLIB=CEE.SCEERUN;
        exec sh -L
fi
```

  A module found in CEE.SCEERUN is loaded from that library into the user's private storage, even if the same module has been put into the LPA. This action can become a concern if the STEPLIB points to a Language Environment runtime library, because several loads are done for each exec() to initialize the environment. If you have a number of users accessing this load library, you can avoid directory I/O as well as I/O to load frequently used members by caching the library in LLA and VLF.

Checking that the sticky bit is set

The z/OS shell is shipped with the sticky bit set on, which reduces I/O and improves performance. Check to see that the sticky bit is still on by issuing:

```
ls -l /bin/sh
```

The first part of the output should be:

```
-rwxr-xr-t
```

The `t` indicates that the sticky bit is on.

Organizing file systems to improve performance

How well the file system performs depends on how it is organized. Because a mountable file system must reside on a single DASD volume, several file systems on a volume or too much activity in a single file system can cause DASD I/O response time to be a bottleneck. In addition, some file system locking

is done on a mountable file system basis. For these reasons, each user should normally have a unique mountable file system.

Another consideration is the placement of files in the file system hierarchy. Files deep in the hierarchy require several lookups each time they are opened.

Improving performance of security checking

To improve the performance of security checking done for z/OS UNIX, define the BPX.SAFFASTPATH FACILITY class profile. This reduces overhead when doing z/OS UNIX security checks for a wide variety of operations. These include file access checking, IPC access checking, and process ownership checking. For more information about the BPX.SAFFASTPATH profile, see “Fastpath support for System Authorization Facility (SAF) ” on page 288.

OMVS command and TSO/E response time

When a user goes into the shell environment using the OMVS command from TSO/E, very long TSO/E response times (several seconds) might be recorded. This can affect the workload management (WLM) goals for TSO users that are based on response time.

Normally, a TSO/E transaction starts when a user enters a command and ends when the command is completed. After the TSO/E command completes, a TGET WAIT is issued, indicating that the current transaction has completed and a new transaction will start when there is more work to be done.

In the OMVS shell environment, however, things work a little differently. A transaction starts when a command is issued from the terminal. After the command is issued, polling is done to wait for output to return from the command. Every half second, there is a test for output and a test (TGET NOWAIT) for terminal input. This goes on for 20 seconds before the session goes into INPUT mode and does a TGET WAIT for terminal input only. TGET NOWAIT does not end the current transaction unless terminal input is found. If there is no more terminal input for over 20 seconds, the transaction does not end until the TGET WAIT is issued and the session goes into INPUT mode.

In effect, TSO/E users in the shell environment can experience response times of up to 20 seconds, often with little service consumption. Response times under 20 seconds occur only when users immediately enter the next command.

Chapter 21. Setting up for sockets

A *socket* is a method of communication between two processes that allows communication in two directions, in contrast to pipes, which allow communication in one direction. The processes using a socket can be on the same system or on different systems in the same network. A program creates a socket with the socket() function.

When setting up for sockets, your two choices are INET and CINET (Common INET). These are network sockets, and this topic describes those sockets in detail. Local network sockets (AF_UNIX), which do not have network connectivity, are also available, but are not discussed. INET and CINET are file systems that are in the AF_INET and AF_INET6 family of sockets. This topic helps you decide which file system is best for you to use and describes how to set it up. It contains examples that are based on an assumed sample configuration. You will need to modify the examples based on the requirements for your installation.

User-written socket applications can use TCP/IP as a communication vehicle. TCP/IP is also the transport provider when users rlogin or telnet from a UNIX workstation directly into the z/OS shell.

You can use CINET configured with just one stack, but this configuration will not run as efficiently as INET. "Choosing between INET or CINET" on page 378 provides background information that you may need.

List of subtasks

Subtask	Associated procedure
Customizing BPXPRMxx for CINET systems	"Steps for customizing BPXPRMxx for CINET" on page 381

Using single stacks

In a single stack environment, the socket application program is always associated with the single TCP/IP stack. Figure 40 on page 377 shows an example of a z/OS UNIX system that uses a single stack.

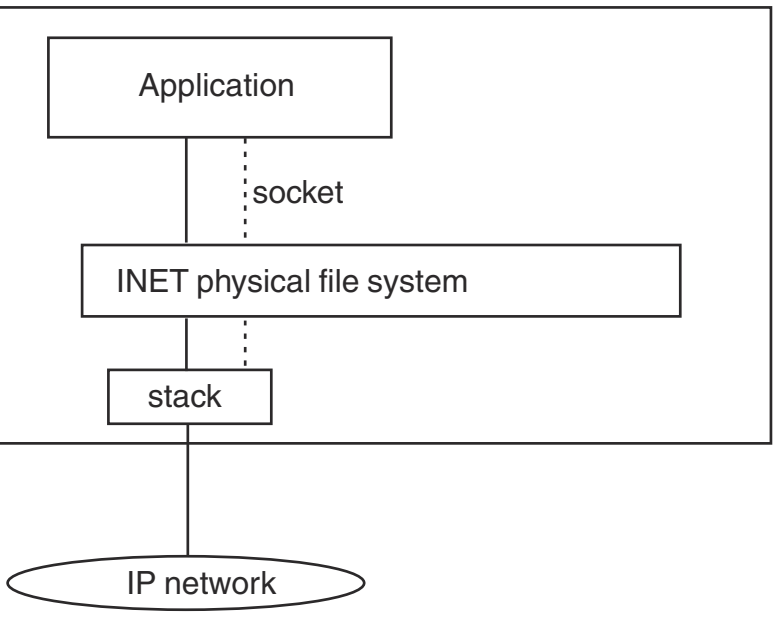


Figure 40. A z/OS UNIX system using a single stack

The TYPE(INET) parameter on the FILESYSTYPE statement defines INET.

INET is defined by the presence of a FILESYSTYPE statement for a socket file system whose ENTRYPOINT is not BPXTCINT. By convention, and in this topic, TYPE(INET) parameter for INET configurations is specified. The entry point for INET is typically EZBPFINI, for z/OS Communications Server (TCP/IP Services).

Using multiple stacks

In the z/OS UNIX Common INET (CINET) environment, the application is associated with multiple TCP/IP stacks unless the application specifically associates itself with a particular stack using the socket call setibmopt(). Figure 41 on page 378 shows an example of a z/OS UNIX system that uses multiple stacks.

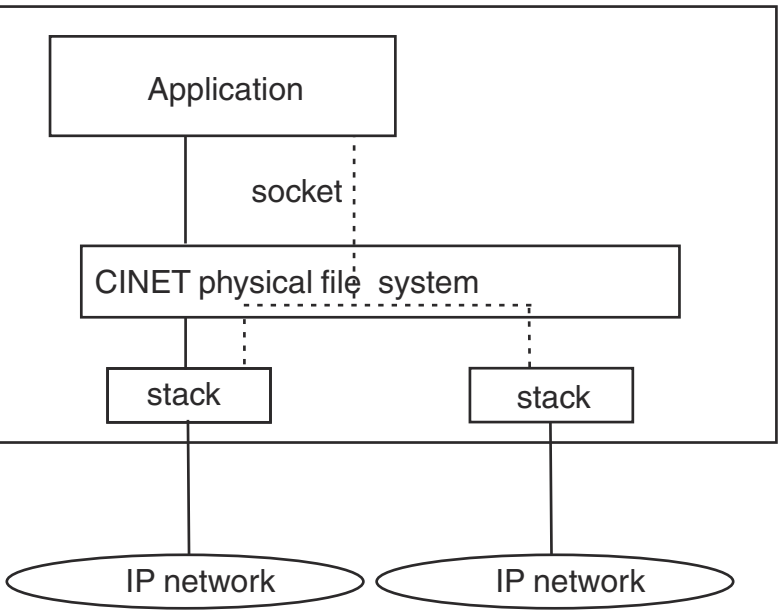


Figure 41. A z/OS UNIX system using multiple stacks.

CINET is defined by the presence of a FILESYSTYPE statement with an ENTRYPOINT of BPXTCINT and the presence of one or more SUBFILESYSTYPE statements that define the actual TCP/IP stacks to be used. By convention, and in this topic, TYPE(CINET) parameter for CINET configurations is used. The entrypoints for the SUBFILESYSTYPE statements are typically EZBPFINI for several instances of the daemon, which is used to route messages, is shipped with (TCP/IP Services) or the entry point for a conforming TCP/IP stack that is provided by a vendor.

For the example in Figure 41 on page 378, you would have the two SUBFILESYSTYPE statements to define the two stacks shown in the illustration.

Choosing between INET or CINET

Previous enhancements to TCP/IP have reduced the need for multiple TCP/IP stacks. CINET may still be a viable choice if you are isolating access from different networks to the same z/OS UNIX system. An example of such a situation would be internal company networks versus internet access.

Both this topic and *z/OS Communications Server: IP Configuration Guide* contain information about running one TCP/IP stack and multiple TCP/IP stacks

Local INET (LINET) has been retired. SUBFILESYSTYPE ENTRYPOINT(BPXTLINT) was originally supplied as an alternative stack under CINET for a performance enhancement for AF_INET socket sessions between programs on the same system. The performance improvements that have been made in the TCP/IP socket stack in the past have removed the need for LINET. If LINET is started, an informational

message is sent to the system console and LINET will deactivate itself. This will not affect the use of sockets, and you can delete the definition of LINET from your BPXPRMxx member at your convenience.

Setting up for INET

Use INET unless you have a special reason to use CINET.

To use the single transport provider support, see the following example of the statements that should be in the BPXPRMxx member of SYS1.PARMLIB.

```
FILESYSTYPE TYPE(INET)
            ENTRYPOINT(EZBPFINI)
NETWORK DOMAINNAME(AF_INET)
            DOMAINNUMBER(2)
            MAXSOCKETS(64000)
            TYPE(INET)
```

To use the single transport provider support, change the MAXSOCKETS value to 64000.

IBM Communications Server for z/OS supports the AF_INET6 address family, which allows socket applications to use the IPv6 APIs. See *z/OS Communications Server: IPv6 Network and Appl Design Guide* for more information about IPv6 APIs.

If you want to use the single transport provider support with both AF_INET and AF_INET6 address families, the following excerpt shows an example of the statements that should be in the BPXPRMxx member.

```
FILESYSTYPE TYPE(INET)
            ENTRYPOINT(EZBPFINI)
NETWORK DOMAINNAME(AF_INET)
            DOMAINNUMBER(2)
            MAXSOCKETS(64000)
            TYPE(INET)
NETWORK DOMAINNAME(AF_INET6)
            DOMAINNUMBER(19)
            MAXSOCKETS(64000)
            TYPE(INET)
```

You can activate AF_INET6 without recycling z/OS UNIX by adding this NETWORK statement to a running configuration with the SETOMVS RESET=() operator command. Specify a BPXPRMxx member that contains just this one statement. However, the TCP/IP stack would have to be stopped and restarted in order to pick up the new definition. You can specify a separate MAXSOCKETS value for AF_INET6 or default to the value specified for AF_INET. In either case, each family has its own separate maximum.

Setting up for CINET

The CINET support enables an installation to connect up to 32 transport providers to z/OS UNIX. The user of the sockets library does not need to change any code to take advantage of the multiple transports connected to the kernel services. Only a maximum of eight z/OS Communications Server TCP/IP stacks can be concurrently active.

z/OS Communications Server: IP Configuration Reference contains the sample TCP/IP configuration files.

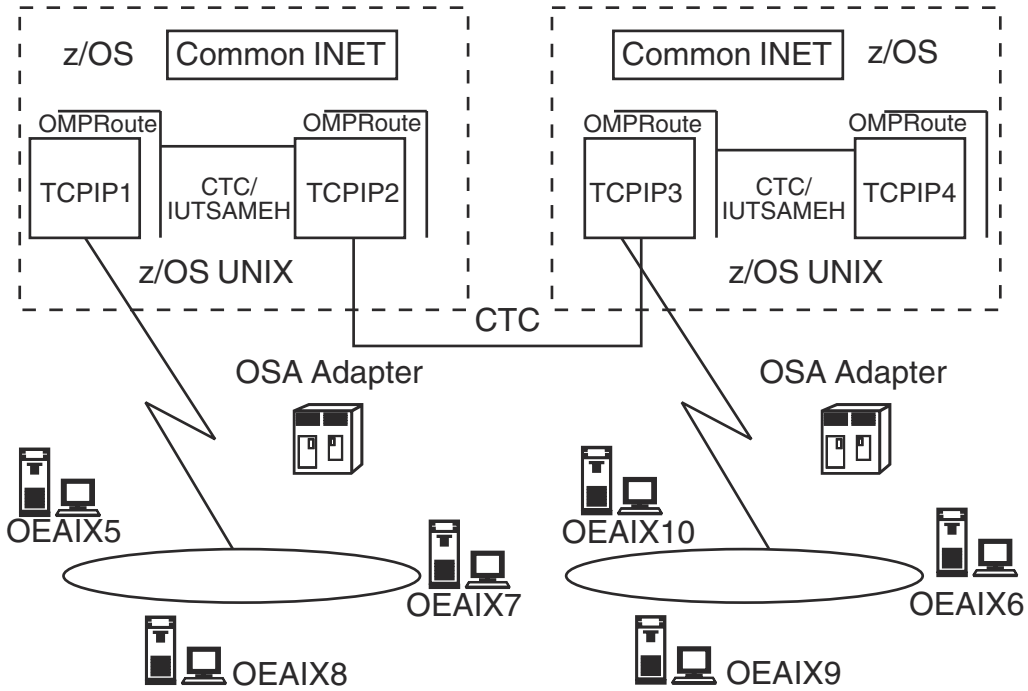


Figure 42. Multiple transport provider support with two z/OS UNIX systems

Supporting multiple transports and providing a single AF_INET or AF_INET6 image to the user means that CINET must perform a set of management and distribution functions that govern how a socket behaves with multiple transports. A fundamental requirement for distributing work across multiple transports is the need to understand the IP configurations of each. The IP configurations are needed to determine which transport should handle a bind(), a connect(), or a sendto() to a particular Internet Protocol (IP) address. An IPv4 address is a 32-bit address defined by the Internet protocols, and an IPv6 address is a 128-bit address defined by the Internet protocols.

When the CINET function processes a socket request that requires it to select only a particular transport based on an input IP address from a user, CINET uses its copy of each transport's IP configuration to select the correct transport to process the user's request. Copies of the IP configurations are maintained by CINET internally and are only used for prerouting a socket call to the correct transport. The transport that is selected performs all of the official transport functions, such as IP routing, once the socket request reaches the transport from CINET.

The internal routing table

Each transport connected to kernel services must provide CINET with a copy of its internal routing table. The CINET function queries the routing tables of the transports connected to the kernel services. After the CINET prerouter function has successfully retrieved and stored routing information from a particular transport, message BPXF206I is issued.

For example, IBM's TCP/IP may refresh its routing tables as part of the OBEYFILE command. Message BPXF207I is issued to the hardcopy log whenever CINET deletes internal routing information for a transport. For example, when a z/OS UNIX-to-TCP/IP connection is severed, the CINET routing information for that TCP/IP is deleted.

You can display the network routing information for all the active transport providers being used by CINET prerouter by using the CINET operand of the DISPLAY OMVS operator command. For example:

```
D OMVS,CINET=ALL
```

Transport providers

The transport providers are specified with the SUBFILESYSTYPE statements in BPXPRMxx or specified with the SETOMVS command. The default transport provider is one of the following:

- The transport provider specified as the default on the SUBFILESYSTYPE statement in BPXPRMxx.
- If DEFAULT was not specified, the transport provider from the first SUBFILESYSTYPE statement will become the default transport provider while it is active.
- If DEFAULT was not specified, the first SUBFILESYSTYPE transport provider specified is not active, and no other stacks are active, then the first transport provider activated becomes the default. The selection of default transport providers can become unpredictable if stacks are stopped and restarted while the specified default transport provider and the first SUBFILESYSTYPE transport provider are inactive.

Limitations of IP configurations using CINET

System programmers and network designers should be aware of the following information about the CINET prerouting function:

Home IP addresses
Two or more transports running on z/OS that connect to z/OS UNIX may contain home IP addresses on the same network or subnetwork. However, load balancing across transports is not done.

Network destinations
Two or more transports may have network destinations that are the same. Again, load balancing across transports is not performed.

Metrics for network routes
All routes are equal and their metrics are compared.

If two or more transports maintain network routes to the same destination network, metric information is needed from each transport in order to correctly select the best route. For IBM's TCP/IP, this is best accomplished when each TCP/IP is running with a dynamic routing daemon (OMPROUTE). When two or more transports maintain indirect routes to the network, statically defined indirect routes (routes to destinations that do not reside on a transport's directly attached links) do not provide adequate metric information to select the shortest route to a destination network.

If two or more transports contain network routes with no metric information or duplicate metrics, then the default transport is called to process the request. The default transport is either the file system that specified DEFAULT on the SUBFILESYSTYPE statement (if active), or it is the first transport that was activated.

Host routes
Host-defined routes are always searched before network routes.

Severed connection to z/OS UNIX services.
If a transport should sever its connection with z/OS UNIX, all routing information for the severed transport is deleted. If the severed transport maintained duplicate home or network routes, these routes are deleted. Subsequent requests for the duplicate routes are routed to the remaining transports.

Customizing BPXPRMxx for CINET

"Steps for customizing BPXPRMxx for CINET" on page 381 shows an example of the statements in the BPXPRMxx member to use the multiple transport provider support.

Steps for customizing BPXPRMxx for CINET

Before you begin, you must know that:

- The names TCPIP1, TCPIP2, TCPIP3, and TCPIP4 are the names of the TCP/IP started tasks. The names must match the job names that are associated with the TCP/IP started task procedure.
- The first TCP/IP has been designated as the default (DEFAULT) transport provider.

- The value specified for the TYPE operand can be any 8-character value, but that value must match on the FILESYSTYPE statement for CINET, on the SUBFILESYSTYPE statements for the transport providers, and on the NETWORK statement for CINET.

Perform the following steps to customize BPXPRMxx for CINET.

1. Specify the following in BPXPRMxx:

```
FILESYSTYPE TYPE(CINET)
            ENTRYPOINT(BPXTCINT)
```

   ---

2. Specify the AF_INET or dual AF_INET/AF_INET6 sockets physical file systems that are to be activated.

```
SUBFILESYSTYPE NAME(TCPIP1)           /* First TCPIP (TCPIP1)  */
               TYPE(CINET)
               ENTRYPOINT(EZBPFINI)
                 DEFAULT

SUBFILESYSTYPE NAME(TCPIP2)           /* Second TCPIP (TCPIP2) */
               TYPE(CINET)
               ENTRYPOINT(EZBPFINI)

SUBFILESYSTYPE NAME(TCPIP3)           /* Third TCPIP (TCPIP3) */
               TYPE(CINET)
               ENTRYPOINT(EZBPFINI)

SUBFILESYSTYPE NAME(TCPIP4)           /* Fourth TCPIP (TCPIP4) */
               TYPE(CINET)
               ENTRYPOINT(EZBPFINI)
NETWORK DOMAINNAME(AF_INET) DOMAINNUMBER(2) MAXSOCKETS(64000)
        TYPE(CINET) INADDRANYPORT(4901) INADDRANYCOUNT(100)
```

When you are done, you have customized BPXPRMxx for CINET sockets processing.

Note:

1. If you want to use IPv6, add the following NETWORK statement:

```
NETWORK TYPE(CINET) DOMAINNAME(AF_INET6) DOMAINNUMBER(19)
```

   IPv6 support is optional.
2. For AF_INET6, you can specify a separate MAXSOCKETS value or let it default to the value specified for AF_INET. In either case, each family has its own separate maximum.
3. For AF_INET6, the INADDRANYPORT and INADDRANYCOUNT keywords are ignored. You can activate AF_INET6 without recycling z/OS UNIX by adding this NETWORK statement to a running configuration with the SETOMVS RESET=() operator command. You would specify a BPXPRMxx member that contains just this one statement. However, you would have to stop and restart each TCP/IP stack in order to pick up the new definition.

After the default transport provider is assigned, the following actions are taken on sockets calls unless transport affinity has been established. (For more information, see “Transport providers” on page 381)

- `getsockname()`: If there is a single transport provider that has been connected or bound to, that transport provider is used. Otherwise, the default transport provider is used.
- `gethostname()` or `gethostid()`: The request is always to be routed to the default transport provider.
- `getsockopt()` or `setsockopt()`: If there is a single transport provider that has been connected or bound to, that transport provider is used. Otherwise, the default transport provider is used.
- Route selection: When an application program makes a request that could be sent to any of a number of transport providers (for example, a datagram sendto() request, connect() request or bind2addrsel() request), the CINET prerouter examines its internal routing table information and decides which transport provider to send the request to. If matching host routes or network routes are found, then following criteria is applied to select the best route in order:

1. If the route is an implicit and NON-DVIPA route, then it is selected to route the request regardless of the interface state (active or inactive).
2. If the route is an implicit and DVIPA route, then it is selected to route the request only if the interface is active.
3. If the route is a non-implicit route (DVIPA or NON-DVIPA), then it is selected to route the request only if the interface is active.
4. If there are multiple matching non implicit routes, then the route with better route type and routing metric is selected to route the request.
5. If no matching host routes are found, the CINET prerouter looks for most specific matching network route.
6. If there are multiple specific matching network routes, then the route with better route type and routing metric is selected to route the request.
7. If there are no matching host or network routes, then active best default route is selected to route the request.

Note:

1. If there is more than one transport provider that maintains a route to the destination with the same route type and routing metrics, then the transport provider that is specified as the default is chosen.
2. When looking for default routes, if more than one transport provider maintains a default route, then the transport provider with the better route type and routing metric is chosen. If there are no active default routes, then the default transport provider is chosen to route the request.

Specifying INADDRANYPORT and INADDRANYCOUNT

These parameters only apply to CINET.

Port reservation information for port 0, INADDR_ANY binds is required for the AF_INET domain with a CINET configuration. Specify this information on the INADDRANYPORT and INADDRANYCOUNT parameters on the NETWORK statement for AF_INET in BPXPRMxx. This also includes the port 0, IN6ADDR_ANY binds for AF_INET6.

If you omit both INADDRANYPORT and INADDRANYCOUNT, then those values will be defaulted. Be careful when allowing the default values because they may not be the values you want. If you do not want to support any reserved ports, then specify INADDRANYPORT(xxx) without specifying INADDRANYCOUNT. In this case, xxx can be any valid numeric value.

INADDRANYPORT specifies the starting port number to be reserved for use by application programs that issue port 0, INADDR_ANY binds. INADDRANYCOUNT specifies how many ports to reserve.

If you are running a CINET configuration and you specify the INADDRANYPORT and INADDRANYCOUNT parameters, you must specify the same values to each transport provider that is specified on the SUBFILESYSTYPE statement.

Refer to the documentation for that transport provider to determine how the port reservation information is specified. For IBM's z/OS Communications Services, use the PORTRANGE profile statement.

If the transport provider does not support the port reservation requirement, you must still specify INADDRANYPORT and INADDRANYCOUNT to process port 0, INADDR_ANY binds. In this case, you should specify a high port number for INADDRANYPORT (for example, 4000) to improve the probability that the port will be available on the transport provider. If the port is not available on any of the transport providers connected to z/OS UNIX, a port 0, INADDR_ANY bind will fail with an ERRNO of EADDRINUSE.

Using specific transports under CINET

The CINET layer performs a multiplexing and demultiplexing function between an application program and the several transports that are active. When a socket is initially created with the socket() call, it is generally available to all the transports. Once the socket becomes associated with a single transport, all subsequent calls go to that one transport; the other transports have no knowledge of the socket at

all. Server sockets typically remain associated with all the transports while client sockets often become associated with just one.

Binding to a specific transport

Each transport under CINET has its own home IP addresses. When a program binds a socket to a specific IP address, that socket becomes associated with the one transport that supports that IP address.

When a program binds to INADDR_ANY or IN6ADDR_ANY, or an IP address of all zeros, the socket remains available to all the transports. This is also true for sockets that are never bound.

Connecting through a specific transport

When a stream, or TCP, socket is connected, it becomes associated with the single transport that is chosen with the best route to the destination IP address specified on the connect() call.

Sockets created from accept() are associated with just the one transport on which the connection arrived.

Requesting transport affinity

When you set *transport affinity*, sockets are restricted to just one transport. A program can associate a socket with a specifically named transport in one of these ways:

1. With setibmopt(IBMTCP_IMAGE), all future socket() calls for AF_INET or AF_INET6 create sockets that are associated with only one specified transport. To invoke them from non-C programs, use BPX1PCT(PC#SetIbmOptCmd). This specification of a specific transport is inherited over fork() and propagated over exec().

   `SetIbmOpt` can be issued more than once to change the chosen transport and affect future sockets that are created. If a blank transport name is used, the process is reset so that no transports are chosen.

   When CINET is not configured, there is only one AF_INET or dual AF_INET/AF_INET6 transport, and all socket() calls for AF_INET or AF_INET6 create sockets with that transport. In this case, setibmopt() has no effect and is ignored.

   If the specific transport is not found, setibmopt() fails with EIBMBADTCPNAME if CINET is installed, and with ENXIO if it isn't.
2. `Call ioctl(SIOCSETRTTD)` associates an existing socket with the one specified transport, removing the others, if any, from the socket.
3. The _BPXK_SETIBMOPT_TRANSPORT environment variable can be set to the name of the desired transport before starting the program. This variable can also be set in the PARM= parameter of a started procedure to have the Language Environment runtime initialization issue a setibmopt() call on behalf of the program being started. This variable can also be included in the _CEE_ENV file.
4. Include a job step that invokes BPXTCAFF. BPXTCAFF is invoked as a job step in front of an existing program in a started procedure or submitted job stream. For example:

   ```
   //STEP0 EXEC PGM=BPXTCAFF,PARM=TPNAME
   //REALSTEP EXEC PGM=MYPGM,PARM='MyParms'
   ```

   The desired transport is specified with the PARM= keyword and must be 1 to 8 uppercase characters. This is the same value that would be specified for _BPXK_SETIBMOPT_TRANSPORT. If PARM= is not supplied, or is blank, then the address space's transport affinity will be reset to no transport selected. This can also be specified as PARM=&VAR, where VAR is a PROC keyword that is passed in from the Start command or is a static system symbol.

   BPXTCAFF sets transport affinity for an address space for the duration of that address space or job. This affinity persists over job steps within the job, persists over UNIX process termination and redubbing, and applies to all UNIX processes running within that address space. BPXTCAFF is intended for use with non-C or POSIX(OFF) programs where the _BPXK_SETIBMOPT_TRANSPORT environment variable is not effective. It is also intended for programs that do not make their own

calls to setibmopt() or BPX1PCT or that cannot be modified to do so. BPXTCAFF exits with one of the following return codes in register 15.

0
Successful. TPNAME matched an AF_INET socket transport.

2
Minor failure. TPNAME did not match any transport but CINET is not configured so transport affinity is moot.

8
Failure. CINET is configured and TPNAME did not match any transport running under CINET.

12
Failure. The interface to the routine was not valid.

To set transport affinity for a TSO address space, you can also invoke BPXTCAFF from TSO/E by issuing:

```
 TSO CALL 'SYS1.LINKLIB(BPXTCAFF)' 'TPNAME'
```

To set transport affinity for an address space using a REXX procedure, you can invoke BPXTCAFF from REXX by issuing:

```
/*  REXX  */
Address Linkmvs BPXTCAFF 'TPNAME'
Exit rc
```

BPXTCAFF makes a call to BPX1PCT(PC#SetIbmOptCmd) with an Arg value of 1 specified to achieve transport affinity for a persistent address space. For more details, see pfsctl (BPX1PCT, BPX4PCT) — Physical file system control in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Restriction: A BPXTCAFF job step must not be used with z/OS UNIX address spaces that are set up for the NFS or DFS clients. The z/OS UNIX services that are needed by BPXTCAFF are not available when the colony is started from the BPXPRMxx member. However, z/OS UNIX cannot finish initialization until the colonies are initialized, so the system will hang. The technique that is described in Step “5” on page 385 sets up affinity for NFS or DFS clients.

5. Use the PARM= parameter of the z/OS UNIX colony address space. To set transport affinity for the NFS Client or DFS Client, use the PARM= keyword of the EXEC statement that starts BPXVCLNY in the colony address space procedure as follows:

   ```
   //MVSCLNT EXEC PGM=BPXVCLNY,TIME=1440,PARM=TP(TPNAME)
   ```

   where the PARM=value is the following:

   - All in uppercase.
   - Starts with "TP(".
   - TPNAME is the left-aligned, 1-to-8-character name of the desired transport.

   If PARM= is specified and does not conform to these rules, the colony is terminated by an EC6 abend with a reason code of 11BE8039. When CINET is configured on the system and the specified transport is not configured under CINET, the colony is terminated by an EC6 abend with a reason code of 11BE803A. In either case, the colony can be restarted after the procedure is corrected by replying to the operator prompt that is issued.

   The _BPXK_SETIBMOPT_TRANSPORT environment variable does not work in a z/OS UNIX colony address space because it does not start under Language Environment.

6. Specify a transport name when a socket is created.

   Individual sockets can be created with transport affinity by passing the transport name directly to the BPX1SOC callable service. This specific socket affinity overrides any process-level affinity that has been established. This feature is not available to C programs through the standard socket() function, but C programs can call BPX1SOC directly. For more information, see in socket or

socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

7. Specify a transport name for GETHOSTID() or GETHOSTNAME().

   A BPX1HST request for GETHOSTID or GETHOSTNAME can be directed to a specific transport by using the BPX1PCT(PC#DIRGETHOST) function. This overrides any process-level affinity that has been established. For more information, see in pfsctl (BPX1PCT, BPX4PCT) — Physical file system control in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Resolver configuration files

The resolver acts on behalf of programs as a client that accesses name servers for name-to-address or address-to-name resolution. The resolver can also be used to provide protocol and services information. To resolve the query for the requesting program, the resolver can access available name servers, use local definitions (for example, `/etc/resolv.conf`, `/etc/hosts`, `/etc/ipnodes`, HOSTS.SITEINFO, HOSTS.ADDRINFO, or ETC.IPNODES), or use a combination of both. How and if the resolver uses name servers is controlled by TCPIP.DATA statements (resolver directives). The resolver address space must be started before any application or TCP/IP stack resolver calls can occur.

This topic explains the format of the resolver data sets and files when they are stored in the z/OS UNIX file system.

Host information

Host information not obtained from a domain name server can be obtained from local host tables. If you want to know what the options are for creating local host tables and how the tables are searched, refer to *z/OS Communications Server: IP Configuration Guide*.

Service information

Figure 43 on page 386 shows an extract of the services file. You can copy the sample service information from `/usr/lpp/tcpip/samples/services` into your `tcpip.ETC.SERVICES` data set or `/etc/services` file.

```
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp
chargen         19/tcp
```

Figure 43. Partial extract of the services information

Protocol information

You can copy the sample protocol information from `/usr/lpp/tcpip/samples/protocol` into your `tcpip.ETC.PROTO` data set or `/etc/protocol` file.

To see the MVS version of the TCPIP.DATA data set, see the sample in the topic on defining TCP/IP client system parameters in TCPCONFIG statement in *z/OS Communications Server: IP Configuration Reference*.

Resolver information

The TCPIP.DATA data set is the only one of the TCP/IP data sets for which a unique copy is needed for each transport provider. This is because the TCPIPJOBNAME statement identifies the TCP/IP address space and the HOSTNAME statement identifies the host name of the TCP/IP address space.

The following example shows a typical TCP/IP syntax:

```
Datasetprefix         TCPIP.TEST1     ; This stack's data set prefix
TCPIPjobname          TCPCO1          ; Stack name
NSinterAddr           127.0.0.1       ; Name server (on this system)
NSportAddr            53              ; Name server port number
ResolveVia            UDP             ; Use UDP for Name server
ResolverTimeout       30              ; 30-second name server timeout;
ResolverUdpRetries    1               ; Retry name server once
HostName              TCPIP1          ; My host name
DomainOrigin          pok.ibm.com     ; My domain origin
Messagecase           mixed           ; Issue mixed-case messages
```

The following example shows a typical z/OS UNIX syntax:

```
domain pok.ibm.com
nameserver 9.114.75.254
nameserver 9.114.171.254
nameserver 9.114.151.254
```

The system processes this information during the initial request for service. It accepts either format for the information supplied regardless of the source selected.

When they are mixed, only the last domain or DOMAINORIGIN data is used and up to 16 name server's addresses are used for initialization. However, if you set up `/etc/resolv.conf` to supply resolver information, you must specify the DATASETPREFIX information in `/etc/resolv.conf` unless you have also set up `/etc/services`, `/etc/protocol`, and `/etc/hosts` files.

Any mix of MVS data sets and z/OS UNIX files can be used. For example, you could use the TCPIP.DATA information from SYS1.TCPPARMS, the service and protocol information from ETC.SERVICE and ETC.PROTO and use the `/etc` directory in the z/OS UNIX file system to record hosts names and addresses of the z/OS UNIX hosts file.

Displaying information about sockets

You can display information about AF_INET, AF_INET6, and AF_UNIX sockets as described in “Displaying information about local and network sockets” on page 278.

Chapter 22. Managing accounting work

You can measure, collect, and report accounting information.

List of subtasks

Subtasks	Associated procedure
Modifying the accounting information for the OMVS and BPXOINIT address spaces	"Steps for modifying accounting information" on page 390
Checking job names and accounting information using IEFUJI	"Steps for activating the IEFUJI exit for OMVS work" on page 392

Using system management facilities (SMF)

To perform accounting for UNIX workloads, use system management facilities (SMF). Basic accounting models that use address-space level data from SMF type 30 records should work correctly for UNIX processes. Be aware that:

- The TCB time in SMF type 30 record includes the time in the kernel address space.
- The address-space level EXCP (I/O) count includes the I/O for UNIX files.
- If the program in a user address space issues fork(), the child inherits the TSO/E user ID and the UID.

To weigh central processor time or I/O or both, use the fields in SMF type 30 records to isolate the resources used. Record type 30 also includes the user identification fields:

- UID
- GID
- Process ID (PID)
- Parent process ID (PPID)
- Process group ID (PGID)
- Session ID (SID)

For detailed file system and file open and close activity data, look in SMF record type 92.

When you perform the accounting, one other major factor to be aware of is that the exec() family of functions typically causes step termination and a new substep is started. The new substep still has the same step number, but the substep number is incremented. Therefore, accounting applications must look for substep_number in addition to job name, job_start_time, and step_number.

The kernel creates other address spaces, such as BPXOINIT, and forks other programs—for example, `/etc/init`. The kernel and all its child processes use the same account number. BPXOINIT is the source of the account number propagated to the `/etc/rc` and daemons.

Because the kernel is a started procedure, you can assign accounting data only by coding a JCL installation exit. Alternatively, you can allow the resources used by the kernel and its forked address spaces to be accounted for as system overhead.

Assigning account numbers for forked address spaces

Account numbers for forked or spawned address spaces are set as follows:

- Forked or spawned address spaces inherit accounting data from the parent address space.
- When daemon processes such as **rlogind** or **cron** create new work using setuid() and exec(), accounting data comes from the user's RACF profile (the WAACCNT value in the WORKATTR segment). If this value is not defined in RACF, the address space will not have accounting data.

  Guideline: The user ID starting the cron daemon should be different than the user ID running the cron jobs in order to pick up the correct WAACCNT information in the WORKATTR segment for the cron jobs that are running.

  In addition, if you have multiple UID(0) users defined, and you want to track SMF accounting data for the cron daemon itself, there is no guarantee you will get the same user ID that started the cron daemon in your SMF type 30 records.

  If there are multiple user IDs in the OMVS segments that share UID(0), it will not be known which RACF profile will be used to create accounting data, so results will be unpredictable.
- Accounting data can also be verified or changed using the IEFUAV or IEFUSI installation exits.

  Restriction: Your IEFUSI exit will not receive any step account information for forked address spaces.
- With the _BPX_ACCT_DATA environment variable, users can change the account data for a process that is about to be exec()'d or spawned.
- With the __spawn() service, the caller can define account information in the spawn inheritance structure.

The IEFUAV exit is only passed control when the IEFUAV exit is activated for subsystem OMVS (or all subsystems). This environment typically describes the environment in which a daemon determines the identity of a client, sets up the security environment, and passes the routine control.

In the case of a fork(), spawn(), or exec() where the accounting data is provided by a superuser, the IEFUAV exit is not passed control.

For forked address spaces, the accounting information cannot be changed from the initial value because it is a single step transaction. The SMF30ACT field is only present in SMF30 subtype records 1 (Job start) and 5 (Job termination). It is not present in SMF30 subtype records 2 (Interval), 3 (Step termination) or 4 (Step total).

Modifying the accounting information for the OMVS and BPXOINIT address spaces

Account information for TSO users who log into the shell environment and run utilities or shell scripts comes from the TSO/E logon panel account field. This is true even for users who have a WORKATTR segment in the security product data base.

Steps for modifying accounting information

Before you begin, you must decide whether to put the IEFJOBS DD statement in the MSTJCLxx member or in the MSTJCLxx module in SYS1.LINKLIB. The advantage to using the MSTJCLxx member is that it is easier to make changes to the master JCL. For more information about MSTJCLxx, see MSTJCLxx in *z/OS MVS Initialization and Tuning Reference*.

Perform the following steps to modify account information by putting the IEFJOBS DD statement in the MSTJCLxx member.

1. Put an IEFJOBS DD statement in the MSTJCLxx member. The statement must point to a data set called SYS1.STCJOBS, which is an FB 80 data set. For example:

```
//MSTJCL01 JOB MSGLEVEL=(1,1),TIME=1440
//         EXEC PGM=IEEMB860,DPRTY=(15,15)
//STCINRDR DD SYSOUT=(A,INTRDR)
//TSOINRDR DD SYSOUT=(A,INTRDR)
//IEFPDSI  DD DSN=SYS1.PROCLIB,DISP=SHR
```

```
//IEFJOBS  DD DSN=SYS1.STCJOBS,DISP=SHR
//IEFPARM  DD DSN=SYS1.PARMLIB,DISP=SHR
//SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
//SYSLBC   DD DSN=SYS1.BRODCAST,DISP=SHR
```

_____

2. Create a data member in the SYS1.STCJOBS data set that has the same name as the started procedure.

If the started procedure is ...	Then the data member should contain ...
OMVS	`//OMVS  JOB  (account data),TIME=NOLIMIT,REGION=0k` `//OMVS  EXEC OMVS`
BPXOINIT	`//BPXOINIT  JOB  (account data),TIME=NOLIMIT,REGION=0K` `//BPXOINIT  EXEC BPXOINIT`

   Account data for BPXOINIT is propagated to the `/etc/init` process (or `/usr/sbin/init`) and all the processes that they create.

_____

3. IPL the system.

_____

When you are done, you have modified the accounting information in both OMVS and BPXOINIT address spaces.

Validating user accounts using the IEFUAV exit

After the IEFUAV exit receives control for forked or spawned address space, the accounting information can be checked.

If the _BPX_ACCT_DATA environment variable and the account data was not specified in the spawn inheritance structure for _spawn only, then the account data passed to the exit is the same as the account data of the parent of the forked/spawned address space.

If the _BPX_ACCT_DATA environment variable or account data was specified in the spawn inheritance structure for _spawn, then this is the account data that is seen in this exit. For spawned address spaces, the account data in the spawn inheritance structure takes precedence over account data from the _BPX_ACCT_DATA environment variable.

If the IEFUAV exit sets a return code indicating that the user is not to be allowed to continue, the reason code 0BFC0432 is issued and the address space is terminated.

- 0BFC0432 — issued from module BPXPRJSR
- Reason code 0432 - JRJsrUavXit - The IEFUAV exit rejected the accounting data

If users are running in the shell, they might see the following message:

```
FSUM9209 cannot execute: reason code 0bfc0432
```

Message BPXP005I is written to the job log for the user:

```
BPXP005I  A fork or  spawn error was encountered.
  Return code 00000070 Reason code 0BFC0432
```

Checking job names and accounting information using the IEFUJI exit

Use the IEFUJI installation exit to check job names, accounting information, or both. You will have to customize your system setup in order to activate the IEFUJI installation exit for z/OS UNIX services. (The OMVS, BPXOINIT and BPXAS address spaces have a subsystem type of STC. Other address spaces that are started by BPXOINIT have a subsystem type of OMVS.)

For more information about the IEFUJI installation exit, see IEFUSI — Step initiation exit in *z/OS MVS Installation Exits*

Steps for activating the IEFUJI exit for OMVS work

Before you begin, you must know that when defining OMVS, you must use the keyword parameter form of the IEFSSNxx member of SYS1.PARMLIB. Subsystems defined in the keyword parameter form can use dynamic SSI services, while positional format cannot.

Perform the following steps to activate the IEFUJI installation exit.

1. Define OMVS as a subsystem by adding it to the IEFSSNxx member.

   For example, to define OMVS, using the keyword parameter form of the IEFSSNxx member:

   ```
   SUBSYS SUBNAME(OMVS)
   ```

   If you define SUBSYS(OMVS) in IEFSSNxx and then use SET OMVS commands, you might receive a warning message stating that the notification of SUBSYS(OMVS) failed. Ignore this message.

   If you want to use the positional parameter form, it is still supported. However, subsytems defined using the positional parameter form cannot use dynamic SSI services. For more information about the positional parameter form of the IEFSSNxx member, see IEFSSNxx in *z/OS MVS Initialization and Tuning Reference* .

   For example, to define OMVS, using the positional parameter form of the IEFSSNxx member:

   ```
   OMVS
   ```

   _____

2. Specify the subsystem type in the SMFPRMxx member. For example:

   ```
   SUBSYS(OMVS,EXITS(IEFUJI))
   ```

   Tip: Specify your installation-specific options for TYPE, INTERVAL, and DETAIL in the SUBSYS statement. If you specify EXITS, then only those listed are invoked for OMVS work. If EXITS is not specified, all SMF exits are invoked.

   _____

3. Specify IEFUJI in the EXITS option of PROGxx. If it is not specified, it will not get control of work attributed to the address spaces (such as logging into the shell environment, running utilities, or executing shell scripts).

   For example, use one of the following:

   - ```
     EXIT ADD EXITNAME(SYSOMVS.IEFUJI) MODNAME(IEFUJI)
     ```
   - ```
     SETPROG EXIT,ADD,EXITNAME=SYSOMVS.IEFUJI,MODNAME=IEFUJI,
      DSNAME=.............,STATE=ACTIVE
     ```

   IEFUJI is called for forked or spawned address spaces, or both.

   _____

4. Add system processes and any daemons you intend to start from `/etc/rc` to the job names exclusion list.

   For example, add the following system processes to the job name exclusion list for the IEFUJI installation exit:

```
ETCINIT
ETCINIT1
ETCINIT2
ETCINIT3
ETCRC
```

When you are done, IEFUJI will have been activated.

Restriction: You cannot distinguish a forked or spawned address space as being used for foreground or background activity. When the IEFUJI installation exit obtains control for a forked or spawned address space, a flag is set in the interface identifying it as a foreground job. In the past, the only time this flag was set was for TSO address spaces. In a TSO address space, there is a TSB pointed to by ASCBTSB. For an address space of subsystype OMVS, ASCBTSB is zero and no TSB exists. Therefore, you cannot count on having a TSB just because the SMF flag identifies it as a foreground job.

If the IEFUJI exit sets a return code indication that the user should not be able to continue, the initiator will try again. An attempt is made to fork the address space again and if the IEFUJI exit sets the same return code, then reason code 0BFC0434 is set and the address space is terminated.

- 0BFC0434 (issued from module BPXPRJSR)
- Reason code 0434 - JrJsrint (Internal error from BPXPRJSR)

If users are running in the shell, they might see the following message:

```
FSUM7726 cannot fork - reason code 0bfc0434
```

Message BPXP005I is written to the job log for the user:

```
BPXP005I  A fork or  spawn error was encountered.
  Return code 00000070 Reason code 0BFC0434
```

Using the IEFUJV job validation exit

At the time of the first fork or spawn, IEFUJV is entered with a subsystem value of OMVS and a job name of BPXYOEJS, which is the job that is used to create the SWA control blocks used by all subsequent forked or spawned address spaces. The IEFUJV exit is not given control for forked or spawned address spaces under subsystem OMVS.

The BPXAS initiators, in which the forked or spawned processes run, do go through the IEFUJV exit with a subsystem value of STC. The IEFUJV exit is given control when the BPXAS initiator is first started. When subsequent fork/spawn requests use that BPXAS initiator address space, the IEFUJV exit is not called because the BPXAS initiator is already active.

The account data that is on the job card for the BPXAS procedure is propagated from the BPXOINIT job. If a BPXOINIT job is defined in SYS1.STCJOBS and the SYS1.STCJOBS data set is set up so that it is used during IPL, then an installation can define account data there that will be propagated to the BPXAS procedure.

The BPXOINIT job is started and the account data is saved in SWA blocks in internal text format. Then, later, when the first BPXAS procedure is started, z/OS UNIX reads the account data that was saved in the SWA blocks and adds the account data to the default job card that it builds. The default job card looks like the following:

```
//BPXYOEJS JOB(acct-data),MSGLEVEL=(0,0),REGION=54M,TIME=60
```

z/OS UNIX takes the account data and reconstructs the internal text to a format that can be placed on a JCL statement. The account data is reconstructed by placing quotes around each account data field and then parentheses around the account data. As a result, the account data might look different than it did

when it was defined on the BPXOINIT job. For example, if BPXOINIT had the account data defined as (AA,BB), then when z/OS UNIX adds it to the BPXYOEJS job card, the account data is reconstructed as ('AA','BB').

Using the IEFUSI step initiation exit

When the IEFUSI exit receives control, one of the parameters that is passed is the region size that was requested for the JOB or EXEC test JCL statement. For a forked address space, this is displayed as 54M. This value comes from the default SWA blocks that are defined in module BPXPRBS. This is the default JCL that is used when creating forked address spaces.

If the IEFUSI exit does not change any of the region limit values, then the region value is propagated from parent to child, overriding the 54M value. If the region limit value is changed, then the region value is not propagated. The value set by the IEFUSI exit is used instead.

If the IEFUSI exit sets a return code indication that the user should not be able to continue, the initiator will try again. An attempt is made to fork the address space again and if the IEFUSI exit sets the same return code, then reason code 0BFC0434 is set and the address space is terminated.

- 0BFC0434 — issued from module BPXPRJSR
- Reason code 0434 - JrJsrint - Internal error from BPXPRJSR

If users are running in the shell, they might see the following message:

```
FSUM7726 cannot fork - reason code 0bfc0434
```

Message BPXP005I is written to the job log for the user:

```
BPXP005I  A fork or  spawn error was encountered.
  Return code 00000070 Reason code 0BFC0434
```

Generating job names for OMVS address spaces

When the kernel provides an address space for a fork or spawn request, the following rules are used when generating the job name:

1. For fork and spawn requests that do not involve changing user IDs, the job name of the child is set to the base job name with a number from 1 to 9 appended at the end. For example, if you logon to TSO, you will have a job name that is the same as your user ID (for example, SMORG). The first fork or spawn creates an address space with SMORG1. In this case, the base job name is SMORG and all children inherit the same base job name.

   Continuing this example, if address space SMORG1 does a fork or spawn, the new child address space will have a job name of SMORG1 to SMORG9. It is possible to have multiple address spaces with the same job name running concurrently.
2. If you run a batch job with a job name that is 8 characters long (such as PAYROLLX), then all child processes created by this job have the same job name.
3. When you use rlogin to enter the system, the rlogin daemon prompts for your user ID and password or password phrase. The daemon then validates the caller and performs a setuid() followed by an exec(). The setuid()/exec() combination triggers the kernel to change the job name of the address space to the user ID. Because rlogin supports 8-character user IDs, any children created by this process will follow the rules defined in Step “1” on page 394 and Step “2” on page 394, depending on the length of the user ID.
4. If a daemon issues a spawn() with user ID, the child address space is assigned a job name that is the same as the user ID.
5. If a daemon does a spawn() or exec() with the _BPX_JOBNAME environment variable set, the address space gets the requested job name.

Any time the job name is changed, the new job name becomes the base job name for future children.

Restriction: JES attributes for a job with a job name assigned with _BPX_JOBNAME cannot be displayed by job display commands under JES2 or JES3.

Chapter 23. IBM Health Checker for z/OS

IBM Health Checker for z/OS checks the current active z/OS and sysplex settings and definitions for an image and compares their values to either those suggested byIBM or defined by you. It identifies potential problems before they affect your availability or, in worst cases, cause outages.

Some of the z/OS UNIX checks are as follows:

USS_AUTOMOUNT_DELAY
Evaluates the delay times of the configuration.

USS_FILESYS_CONFIG
Evaluates the configuration of the file system.

USS_CLIENT_MOUNTS
In a shared file system configuration, checks whether each file system that can be locally accessed by a non-owning system is indeed locally accessed.

USS _HFS_DETECTED
Checks all mounted file systems and issues a message if any file systems of type HFS are found.

USS_PARMLIB_MOUNTS
Verifies that file systems that are specified in the BPXPRMxx member were mounted successfully. It checks the values that were specified in BPXPRMxx at initialization time. If BPXPRMxx was updated after initialization with different values, those changed values are not included in subsequent checks. Only the initial values are checked.

USS_MAXSOCKETS_MAXFILEPROC
Evaluates whether the MAXSOCKETS and MAXFILEPROC statements in the BPXPRMxx member are set high enough.

USS_PARMLIB
Determines whether there are differences between current system settings and the settings defined in the BPXPRMxx member. If a difference is found, an exception message is issued. You will receive a report that lists the differences.

For a complete list of z/OS UNIX checks, see *IBM Health Checker for z/OS User's Guide*.

Chapter 24. Namespaces for z/OS UNIX

In support of z/OS Container Platform technology, z/OS UNIX provides functionality similar to that of Linux® namespaces. Namespaces provide the appearance of isolation for various system resources. To a process within one of these namespaces, only resources local to that specific namespace are visible and can be manipulated. Similarly, changes within a namespace do not affect resources and processes outside of that namespace.

The root (or initial) namespaces represent the global view. By default, all processes belong to these namespaces. Namespace affiliation is inherited from the parent process when new processes are created. Processes running exclusively in the root namespaces should function exactly as they did before namespaces were supported. Root namespaces consist of the user namespace, the mount namespace, the PID namespace, the network namespace, the IPC namespace, and the UTS namespace.

Restriction: z/OS UNIX only supports the creation of PID, IPC, and UTS namespaces by users.

You can use the clone, setns, and unshare callable services to create namespaces and change a process's namespace affiliation. For more information about these services, see clone, setns, and unshare in *z/OS UNIX System Services Programming: Assembler Callable Services Reference.* To invoke these services, users must either be granted READ access to the CONTAINERS resource profile in the UNIXPRIV class or be a superuser. For more information about providing data and system security, see Chapter 4, "Establishing UNIX security," on page 45.

Namespaces persist for the life of all processes within the namespace. They are freed only after the final process is terminated or leaves. For hierarchical namespaces such as the PID namespace, the namespace will continue to persist until all descendant namespaces have ended. Namespaces will also persist even without affiliated processes or descendant namespaces if the corresponding namespace file in the PROC file system is in use.

Mount namespaces

Mount namespaces allow for the isolation of the mounts for all processes within a specified namespace. Mounts and unmounts that are performed within one mount namespace will not be visible from another namespace.

As with the other namespaces, the clone and unshare callable services can be used to create a new mount namespace. These services create a copy of the mounts in the current mount namespace to be used in the new mount namespace. The difference between clone and unshare is that clone preserves the mount propagation of each mount, but unshare does not.

Each mount has an associated propagation type. The propagation type can be specified by using the following flags on a mount call:

MS_PRIVATE
The mount is private and changes in one mount namespace are not reflected in another. This is the default.

MS_UNBINDABLE
This is similar to a private mount but also restricts bind mounts from being performed to this file system.

Restriction: Some commands can only be used while running within the root mount namespace. The **oedit**, **obrowse**, and **oview** commands always execute within the root mount namespace even if they are invoked from within a new mount namespace.

PID namespaces

To give the illusion that each PID namespace is a complete set of processes, PID namespaces provide virtual isolation of processes. The first process in a PID namespace is assigned a PID of 1 within that

namespace and will be the initial process. This initial process must persist for the life of the namespace and will serve as the parent for any orphaned processes. To provide a certain amount of protection from accidental termination, only signals that have a handler defined can be sent to the initial process of a PID namespace. When the initial process undergoes termination, new processes are no longer allowed to enter the namespace. All processes within the PID namespace will receive a SIGKILL signal.

Unlike the other supported namespace types, PID namespaces are hierarchical. Processes within a descendant PID namespace are visible in ancestor PID namespaces and are assigned local process identifiers, which means that a process might be identified by different PIDs depending upon which namespace the process is being referenced within.

Processes cannot change their PID namespace affiliation. Use of the setns or unshare callable services with the CLONE_NEWPID flag specified will instead change the PID namespace that is associated with any children created after the call. The PID namespace of the caller will remain unchanged.

The setns and unshare callable services can introduce processes into a PID namespace that are not descendants of the PID namespace initial process. If orphaned, these processes are reparented to the initial process in the PID namespace of their parent process. For example:

- Process A in the root PID namespace invokes the setns callable service specifying PID namespace X.
- Process A invokes the fork callable service creating process B within PID namespace X.
- Process A terminates.
- Process B is reparented to the initial process in process A's PID namespace, the root PID namespace.

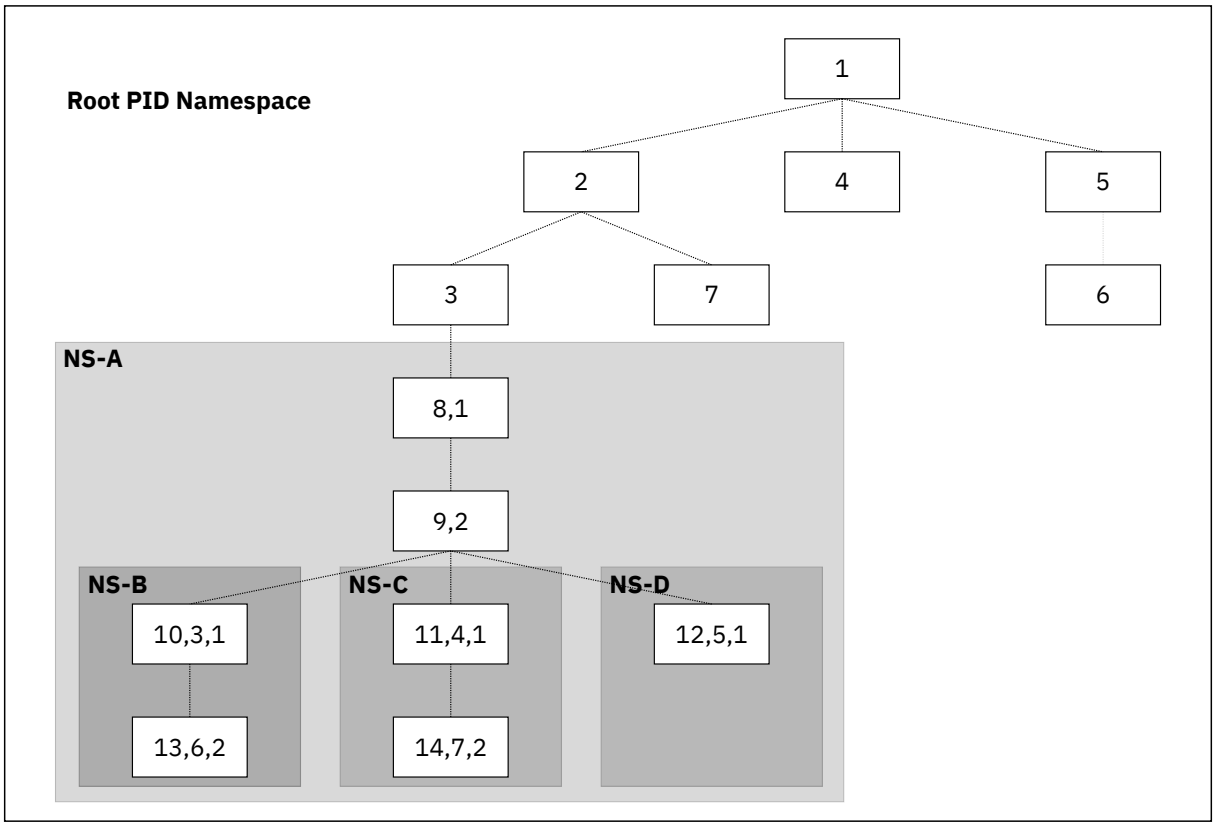
Restriction:

- The maximum number of processes that can be active at the same time in a PID namespace is one half of the current MAXPROCSYS value.
- Any process ID passed as input to a callable service must be a process ID that exists within the caller’s PID namespace. Similarly, process IDs returned as output by a callable service are from within the context of the caller's PID namespace.

Nesting PID namespaces

The following figure illustrates the assignment of local process identifiers within nested PID namespaces.

- Processes 1 through 7 are only in the root PID namespace and are referenced by a single identifier, which is their global or root PID.
- Processes 8 and 9 are within PID namespace A. In the root PID namespace they are identified as PIDs 8 and 9, but they appear as PIDs 1 and 2 within PID namespace A. Process 8 serves as the initial process within PID namespace A.
- PID namespaces B, C, and D are nested within PID namespace A. Processes that are created within these PID namespaces will have a root PID, a PID in namespace A, and a PID within their local PID namespace.



Restriction: The maximum nesting depth for PID namespaces is limited to five (including the root PID namespace).

Shared file system environment

In a shared file system environment, a PID namespace is only valid on the current system. PID namespaces are not viewable by other systems within the same SYSBPX sysplex group.

IPC namespaces

IPC namespaces provide virtual isolation of the interprocess communication (IPC) resources. These resources include message queues, semaphores, and shared memory. IPC namespaces give the appearance that each IPC namespace is a complete set of IPC services and cannot interact with IPC services in other IPC namespaces. An IPC identifier (message queue identifier, semaphore identifier, or shared memory identifier) created within an IPC namespace is only meaningful when used within that specific namespace. It can have the same value as an IPC identifier in another namespace, but it will refer to a different functional instance. For example, a message queue identifier created in one IPC namespace may have the same value as a message queue identifier in another namespace, but the processes cannot use those identifiers to send messages to one another. Similarly, the **ipcs** shell command will only display the IPC functional instance information that is relevant to the current IPC namespace.

Restriction: The maximum number of IPC resource identifiers available in a user-created IPC namespace is one half of the total available to the system.

UTS namespaces

UTS namespaces provide virtual isolation of the system hostname and domain name. You can change these values within a UTS namespace without affecting processes outside the namespace. This functionality is primarily implemented by IBM z/OS Communications Server.

Restriction: To use UTS namespace functionality, IBM z/OS Communications Server must be configured and available on the system.

Restriction: If you are using common INET (CINET), you must set transport affinity before UTS namespaces can be created. For more information about transport affinity, see "Requesting transport affinity" on page 384.

Appendix A. Commonly used environment variables

An *environment variable* is a variable that describes the operating environment of a process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone. Setting an environment variable is optional. If a variable is not set, it will not have any value. This topic contains a partial list of _BPX, _BPXK, and _CEE environment variables.

For information about environment variables used by a particular command, read the description of that command. For more information about environment variables that are used by the C-RTL, see Environment variables specific to the z/OS XL C/C++ library in *z/OS XL C/C++ Programming Guide*.

_BPX environment variables

This section contains a partial list of the _BPX environment variables.

_BPX_ACCT_DATA
Used by the exec callable service to change the account data of the new process image. For the rules on specifying account data, see exec (BPX1EXC, BPX4EXC) — Run a program in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

_BPX_BATCH_SPAWN
Specifies whether BPXBATCH is to use spawn instead of either fork or exec when executing programs.

SPAWN
Spawn is used to execute the program. Data definitions are carried over into the spawned process.

Guideline: When using _BPX_BATCH_SPAWN, you should consider two other environment variables that affect spawn behavior, _BPX_SHAREAS and BPX_SPAWN_SCRIPT. For more information, see BPXBATCH - Run shell commands, shell scripts, or executable filesBPXBATCH in *z/OS UNIX System Services Command Reference*.

NO
Either fork or exec is used to execute the program. NO is equivalent to the default.

_BPX_BATCH_UMASK
Modifies the permission bits on newly created files instead of using the default mask, if PGM has been defined. For more details, see BPXBATCH - Run shell commands, shell scripts, or executable filesBPXBATCH in *z/OS UNIX System Services Command Reference*.

_BPX_CONTAINER_ID
Used by the exec callable service to set the container ID of the new process image. For the rules on specifying a container ID, see the usage notes in exec (BPX1EXC, BPX4EXC) — Run a program in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

_BPX_CONTAINER_POD_ID
Used by the exec callable service to set the pod ID of the new process image. For the rules on specifying a pod ID, see the usage notes in exec (BPX1EXC, BPX4EXC) — Run a program in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

_BPX_CONTAINER_QUAL
Used by the exec callable service to set the container qualifier of the new process image. For the rules on setting the content qualifier, see the usage notes in exec (BPX1EXC, BPX4EXC) — Run a program in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

_BPX_JOBNAME
Specifies the job name of the process, or changes the name. You can specify a string of 1-to-8 alphanumeric characters. Incorrect specifications are ignored. The environment variable is processed on the next spawn() or exec() service with the specified job name being used in the new image.

Restriction: Before the job name can be changed, the invoker must have appropriate privileges. The privileges include either superuser authority or READ permission to BPX.JOBNAME FACILITY class

profile. The invoker must also be running in a space created by the fork callable services. Otherwise _BPX_JOBNAME is ignored.

_BPX_PTRACE_ATTACH
Used when you want to debug target programs.

YES
Programs that are invoked by the spawn, exec, and attach_exec callable services or by the C language spawn() and exec() functions are loaded into user-modifiable storage. Then those target programs can be debugged. The programs that are loaded into storage when the target program is executed, except for modules loaded from LPA, are also loaded.

_BPX_SHAREAS
Specifies whether the spawned child process is to be run in a separate address space from the login shell's address space or in the same address space. Use _BPX_SHAREAS is to improve performance in the z/OS shell. The spawn callable service uses _BPX_SHAREAS when creating child processes.

Restriction: If tcsh is your login shell, do not use BPX_SHAREAS.

YES
The child process is created on a subtask in the parent's address space. If the request cannot be honored, the child is created in another address space.

NO
The child process is created in a new address space. NO is the default.

MUST
The child process is created on a subtask in the parent's address space. If the request cannot be honored, the request will not complete.

Restriction: Sometimes the YES and MUST values cannot be used. For more information, see spawn in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

For a discussion of the benefits and side effects of using BPX_SHAREAS, see "Setting _BPX_SHAREAS and _BPX_SPAWN_SCRIPT" on page 373.

_BPX_SPAWN_SCRIPT
Indicates whether the specified file is to be treated as a shell script. Use _BPX_SPAWN_SCRIPT to improve performance when running z/OS shell scripts.

YES
Treats the specified file as a shell script if it is not an executable process image file or a REXX exec. The shell is also executed to run the specified shell script. You can specify the path name for the shell in the SHELL environment variable on spawn's environment data list. You can also use `/bin/sh` as the default. The first argument passed when spawning a shell script should be the path name of the shell script.

NO
If the specified file is not an executable process image file or a REXX exec, the spawn callable service fails with the ENOEXEC return code. NO is the default if _BPX_SPAWN_SCRIPT is not specified or if it contains an unsupported value.

For more information about performance considerations when using _BPX_SPAWN_SCRIPT, see "Setting _BPX_SHAREAS and _BPX_SPAWN_SCRIPT" on page 373.

Restriction: If tcsh is your login shell, do not use _BPX_SPAWN_SCRIPT because it is only used for improving performance of `/bin/sh` scripts.

_BPX_TERMPATH
Enables shell scripts to determine if the user logged on from TSO, rather than from rlogin or telnet.

_BPX_UNLIMITED_OUTPUT
Specifies output limits that are processed only for non-local spawn requests.

Restriction: To use _BPX_UNLIMITED_OUTPUT, the caller must be a superuser or be permitted to the BPX.UNLIMITED.OUTPUT FACILITY class profile with READ access or greater.

YES
Specifies unlimited spooled output.

NO
Specifies that the default spooled output limits is to be used. Not defining or specifying a value is the equivalent of specifying NO and the defaults limits are not overridden.

_BPX_USERID
Specifies that the child process is to be created with the specified MVS user identity. _BPX_USERID is used by the spawn callable service.

Example: _BPX_USERID=DANIEL creates a child process with the DANIEL user ID to run the spawned program. Authorization for use of the _BPX_USERID is the same as that for the setuid() function. Child processes running with a different user identity than the parent's are always created in a new address space. _BPX_SHAREAS is ignored in this case.

Using _BPX_USERID can improve performance for a program. The program can establish a new user identity for the child that runs the spawned program, instead of creating a child with the original user identity and having the child establish a new user identity. Otherwise, that program would have forked a new address space before establishing a new user identity for the new address space by issuing initgroups(), setgid(), and setuid(), and so forth. Then it would have done an exec() of the program that was to run under the new user identity.

Be careful when using _BPX_USERID. Typically, environment variables passed on spawn are still active in the child process. If _BPX_USERID is set in the parent and not cleared in the child, any spawn calls issued by the child picks up the same _BPX_USERID setting. This behavior is likely to be undesirable. The support that allows the specification of a user ID in the inheritance structure on spawn does not have this drawback.

_BPXK environment variables

A partial list of the _BPXK environment variables is as follows.

_BPXK_AUTOCVT
Used when automatic conversion of tagged files from ASCII to EBCDIC and from EBCDIC to ASCII is enabled. When set, this variable overrides the AUTOCVT setting in BPXPRMxx.

For fork (BPX1FRK/BPX4FRK), spawn (BPX1SPn/BPX4SPN), exec (BPX1EXC/BPX4EXC), and pthread_create (BPX1PTC/BPX4PTC), _BPXK_AUTOCVT is propagated from the parent to the child. For pthread_create, the parent is the Initial Program Task (IPT).

For more information about converting files between code pages, see Chapter 11, “Converting files between code pages,” on page 249.

ON
Activates the automatic file conversion of tagged files from ASCII to EBCDIC and from EBCDIC to ASCII, using Enhanced ASCII. This option affects conversion for I/O for regular, pipe, and character-special files that are tagged. For more information about Enhanced ASCII, see “Enhanced ASCII” on page 249.

OFF
Deactivates the automatic file conversion of tagged files. OFF is the default.

ALL
Activates the automatic conversion of tagged files that are supported by Unicode Services. This option affects conversion for I/O for regular and pipe files that are tagged. Setting or unsetting ALL has no effect after conversion for a file begins. If the conversion is between EBCDIC and ASCII, this option also affects conversion for I/O for character special files.

_BPXK_CCSIDS
Defines an EBCDIC/ASCII pair of valid coded character set IDs (CCSIDs) to be used when automatically converting data or tagging new files. For example:

```
_BPXK_CCSIDS=(1234,5678)
```

_BPXK_DAEMON_ATTACH
Attaches the security environment of the caller of either the setuid(), seteuid(), or setreuid() services to the security environment of the target UID. This combined security environment is called a nested ACEE. With the nested ACEE, the new client identity can then access RACF delegated resources to which only the caller (for example, a daemon), but not the client, is permitted. A RACF-protected resource is delegated if the profile protecting the resource has the string `'RACF-DELEGATED'` in the application data (APPLDATA) field. For more information, see Defining delegated resources in *z/OS Security Server RACF Security Administrator's Guide*.

YES
Attaches the security environment of the caller of either the setuid(), seteuid(), or setreuid() services to the security environment of the target UID. The new client identity can then access MVS resources that are protected by profiles with

```
APPLDATA('RACF-DELEGATED')
```

NO
Security environments are built the way that they normally are. NO is the default.

_BPXK_DISABLE_SHLIB
Specifies whether normal system shared library program processing is enabled (NO) or disabled (YES) for a process.

YES
System shared libraries are disabled. When loading a program with the system shared library extended attribute (`st_sharelib`), the attribute is ignored and the program is loaded into the caller's private storage. Virtual storage is not allocated from the caller's region for system shared libraries.

NO
System shared libraries are enabled, with normal processing of system shared library programs. NO is the default setting.

Note:

1. The scope of _BPXK_DISABLE_SHLIB scope is process-wide.
2. In a multiple-process address space, each process must disable system shared libraries to prevent SHRLIBREGSIZE bytes from being allocated from the caller's region. All processes in the same address space share the region.
3. The _BPXK_DISABLE_SHLIB setting is propagated on both fork and spawn from the parent to the child process.
4. The z/OS UNIX spawn (BPX1SPN), exec (BPX1EXC and BPX1ATX) and oe_env_np (BPX1ENV) services have support for _BPXK_DISABLE_SHLIB.
5. The use of the _BPXK_DISABLE_SHLIB environment variable disables the sharing of storage for system shared libraries and can increase the amount of real or auxiliary storage consumed.

_BPXK_FORCE_CANCEL
Controls the behavior of the pthread_cancel() service for the scope of the process that sets this environment variable.

YES
Specifies that the pthread_cancel() service is to cancel the target thread even if it is not in a normally cancelable state. To accomplish this task, the pthread_cancel() service can wait up to three seconds to ensure that the targeted thread is terminated. If the normal signal mechanism does not terminate the thread in that time, then the nonresponding thread is terminated with a 422 ABEND, reason code 1A0. The threads that can be canceled with this setting include:

- Threads that are waiting but do not use a z/OS UNIX service to enter the wait.

- Threads that are running in an MVS service other than one provided by z/OS UNIX.
- Threads that are running with a non-z/OS UNIX linkage stack entry at the top of the stack.

This environment variable does not override the interruptability state set by pthread_setintr() or the interruptability type set by pthread_setintrtype(). The behavior for a thread that was created but not yet assigned to a task control block (TCB) is not affected by this environment variable. The cancel remains pending and is delivered when the thread is assigned to a TCB. Threads that are waiting in a z/OS UNIX service that is not defined as a cancellation point are also not affected by this environment variable. See the usage notes in pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state in *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for the definition of thread cancellation points.

NO
Specifies that the pthread_cancel() service retains the existing behavior. NO is the default.

_BPXK_GPSENT_SECURITY
The w_getpsent() service (BPX1GTH/BPX4GTH) can use the thread-level identity that is created by the pthread_security_np() service when it checks the ownership of process information.

THREAD
Any thread that is running in the invoking process uses the thread-level identity that is created by pthread_security_np(), if any exists, when it processes the w_getpsent() service. Task-level identities that are not created by pthread_security_np() remains ignored by w_getpsent().

PROCESS
w_getpsent() ignores the thread-level identity and uses the process-level identity. PROCESS is the default.

_BPXK_INITTAB_RESPAWN
Specifies whether a process is to be dynamically started with the respawn attribute.

YES
Specifies that a process is to be started with the respawn attribute. Setting the YES attribute after the process has started does not affect the setting of the respawn attribute. If a process is started by a spawn with _BPXK_INITTAB_RESPAWN=YES (set by an export shell command, for example), the shell invokes the target program. The program will be automatically restarted when it ends, even if it was not originally started from the `/etc/inittab` file.

To set the variable to YES, you must have superuser authority.

NO
Disables the respawn capability of the process.

The NO setting must be set by an application (by a putenv call, for example), while it is running.

You can choose the NO setting to allow for a problem to be fixed if one forced the application to end. Doing so prevents the application from being restarted automatically again with the same problem.

_BPXK_JOBLOG
Specifies whether WTO messages are to be written to an open job log file. For more information about setting _BPXK_JOBLOG, see "Writing messages to a job log file" on page 290.

nn
The job log messages are to be written to open file descriptor *nn*.

NONE
Job log messages are not written. NONE is the default.

STDERR
Messages are written to the standard error file descriptor, 2.

_BPXK_MDUMP
Specifies whether a SYSMDUMP is to be written to the current working directory or an MVS data set.

OFF
The dump is written to the current working directory. OFF is the default.

This dump is written only if the user allocates a SYSMDUMP data set for the TSO/E session. The system creates a file in the user's working directory, names it coredump.pid, where *pid* is the process ID for the process that is being dumped, and writes the core dump (SYSMDUMP) in hexadecimal format.

MVS data set name
The dump is written to an MVS data set. The data set name can be up to 44 characters long. It can also be uppercase or lowercase. If it is lowercase or mixed case, the data set name is folded to uppercase.

The data set name must be a fully qualified name. It must also be preallocated and cataloged.

z/OS UNIX name
The dump is written to a z/OS UNIX file. The file name can be up to 1024 characters long.

The file name must be an absolute path name; that is, it must begin with a slash. The slash refers to the root directory, and the file is created in that directory.

_BPXK_PCCSID
Identifies the program CCSID for the running thread or user. It can be used to override the internal default of 1047 (EBCDIC). Any value between 0 and 65535 can be assigned, but to avoid any subsequent errors, only values that are supported by Unicode Services should be used. Setting or unsetting this variable has no effect after conversion for a file begins. When unset, the internal value of the program CCSID reverts to the default of 1047.

_BPXK_SETIBMOPT_TRANSPORT
Used in a Common INET configuration to choose a socket stack for a program.

_BPXK_SUID_FORK
Specifies whether the setuid indicator is propagated to a child address space that was created by a fork process.

YES
The setuid indicator is propagated to child processes that were created by a fork process. If those children perform a job step exec, it is treated as a new job exec. A new job exec updates job-related attributes to match the RACF identity.

NO
The setuid indicator is not propagated to child processes that were created by a fork process. A job-step exec from the child will not be a new job exec. NO is the default.

_BPXK_TECHNIQUE
Specifies the Unicode Services conversion technique to use for the I/O conversion operation. Setting or unsetting this variable has no effect after conversion of the file starts. If _BPXK_TECHNIQUE is not specified, the default is LMREC.

R
Roundtrip

E
Enforced subset

C
Customized subset

L
Language Environment behavior

M
Modified for special use

0-9
User-defined conversions

_BPXK_TIMEOUT
Specifies whether the process should time out or not.

SMF
Uses the JWT|TWT|SWT values specified in SMFRMxx.

NONE
Specifies that this process is not to be timed out.

The timeout value that is used is the SMF settings for JWT/TWT/SWT. The _BPXK_TIMEOUT variable is ignored when PWT is set to SMF. It is honored when BPXPRMxx PWT is set to ENV or SMFENV.

_BPXK_UNICODE_MAL
Specifies the Unicode Services substitution action to take for the conversion operation when a source character is malformed. Setting or unsetting this variable has no effect after conversion for a file begins. _BPXK_UNICODE_SUB must be YES for this option to apply. F

NO
Unicode Services performs the substitution action. NO is the default.

YES
Unicode Services and, therefore z/OS UNIX, terminates the I/O with an error. _BPXK_UNICODE_SUB must be YES for this option to apply.

_BPXK_UNICODE_SUB
Specifies the Unicode Services substitution action to take for I/O conversion operation when a source character is not convertible to a target character. Setting or unsetting this variable has no effect after conversion for a file begins.

NO
Instructs Unicode Services and, therefore z/OS UNIX, to terminate the I/O with an error. NO is the default.

YES
Instructs Unicode Services to place a substitute character in the output buffer.

_BPXK_UNICODE_TECHNIQUE
Specifies the Unicode Services conversion technique to use for the I/O conversion operation. Setting or unsetting this variable has no effect after conversion for a file starts. If _BPXK_UNICODE_TECHNIQUE is not specified, the Unicode Services default applies. Any eight of the following characters techniques can be specified. Do not specify spaces or commas.

R
Roundtrip

E
Enforced subset

C
Customized subset

L
Language Environment behavior

M
Modified for special use

0-9
User-defined conversions

_BPXK_UNUSEDTASKS
Specifies whether to keep pthreads in a wait.

KEEP
Do not limit the number of pthreads in a wait.

TERM
Terminates waiting pthreads after 30 seconds. TERM is the default.

_BPXK_WLM_PROPAGATE
Controls propagation of WLM enclaves that the z/OS UNIX kernel did not create.

YES
z/OS UNIX services propagate all enclaves (address spaces or task). The services that propagate enclaves are fork(), exec(), spawn(), pthread_create(), _osenv(), and FastCGI. YES is the default.

NO
z/OS UNIX services do not propagate any enclaves that the z/OS UNIX kernel did not create.

_CEE environment variables

This section contains a partial list of the _CEE environment variables. For more information about these environment variables, see Environment variables specific to the z/OS XL C/C++ library in *z/OS XL C/C++ Programming Guide*.

_CEE_ENVFILE
Enables a list of environment variables to be set from a specified file. It does not strip trailing white spaces from each name=value read from a file.

_CEE_ENVFILE must be set through the ENVAR runtime option during initialization of a parent program.

_CEE_ENVFILE is intended for use in batch programs. Because it can supersede built-in shell environment variables such as LIBPATH, do not use _CEE_ENVFILE in the shell environment.

When both _CEE_ENVFILE and _CEE_ENVFILE_S are specified, _CEE_ENVFILE_S takes precedence.

_CEE_ENVFILE_S
Enables a list of environment variables to be set from a specified file. It strips trailing white spaces from each name=value line read from a file.

_CEE_ENVFILE must be set through the ENVAR runtime option during initialization of a parent program.

When both _CEE_ENVFILE and _CEE_ENVFILE_S are specified, _CEE_ENVFILE_S takes precedence.

Appendix B. Modules for the login and logout functions

FOMTLINP module for the login function

The FOMTLINP module (`/bin/fomtlinp`) performs the `login` function. The `login` command initiates sessions on the system for the user that is specified by the user parameter. You can also specify environment variables to be added to the user's environment.

The FOMTLINP module is the interface that is used by rlogin and telnet in z/OS UNIX. In UNIX programs, **rlogin** calls the **login** command; for the z/OS shell, **rlogin** calls this module. For z/OS UNIX, **rlogin** checks passwords and password phrases.

```
*********************************************************************
*
* Function:
* --------
*
*   This routine is attach_exec()ed to or spawn()ed to from a
*   non-superuser caller (unless UID 0 is logging on).
*
*   It receives an open manager and subsidiary pseudo-TTY pair as input.
*   It sets up file descriptors 0/1/2 as usual, sets up several
*   environment variables, fork/exec()s /bin/fomtlinc to do the utmpx
*   recording, and than exec()s to the shell.
*
* Parameters:
* ----------
*
*   1:IN      argc -- usual main() parameter
*
*              = 17 -- normal version
*
*
*
*   2:IN      argv -- usual main() parameter
*
*               note: all arguments are the usual NULL-terminated C/370
*                     strings
*
*          max
*          len  argument   description
*          ---  --------   ------------------------------------------
*
*           15  argv[0]  = program name
*
*                          "fomtlinp"
*
*
*           16  argv[1]  = magic number string (to prevent accidental
*                          invocation from shell command line)
*
*                          "*4OurhrEa)R0,H/h"   (required value)
*           47  argv[2]  = message catalog name for catopen()
*
*                          Empty string means use the default message
*                          catalog ="fomcmcat.cat".
*
*                          catopen() will supply the full path
*                          name by looking at any inherited settings
*                          for LC_MESSAGES, NLSPATH, etc. (note that
*                          catopen() is issued with the NL_CAT_LOCALE
*                          parameter.)
*
*           --  argv[3]  = message catalog set for catopen()
*
*                          "0" means to use the default catalog set,
*                          which is 2
*
```

```
*
*           --  argv[4]  = manager pseudo-TTY file descriptor
*
*                          The correct value is required, if the manager
*                          TTY file descriptor is open in the
*                          spawned process.  If the manager TTY is
*                          closed (perhaps because FD_CLOEXEC was set),
*                          this parameter must be the number of some
*                          closed file descriptor.
*
*
*           --  argv[5]  = subsidiary pseudo-TTY file descriptor
*
*                          (correct value is required -- must be open)
*
*
*           --  argv[6]  = highest used file descriptor
*
*                          (This value is used only if fcntl(F_CLOSFD)
*                          fails (perhaps because one of the file
*                          descriptors was opened by an authorized
*                          program, etc.).  fomtlinp will then close
*                          (one-by-one) all file descriptors from 3 to
*                          argv[6] + argv[7] onclusive.)
*
*                          This argument should not be needed by anyone
*                          other than the TSO/E OMVS command.
*           --  argv[7]  = extra file descriptors to close
*
*                          (This value is used only if fcntl(F_CLOSFD)
*                          fails (perhaps because one of the file
*                          descriptors was opened by an authorized
*                          program, etc.).  fomtlinp will then close
*                          (one-by-one) all file descriptors from 3 to
*                          argv[6] + argv[7] inclusive.)
*
*                          This argument should not be needed by anyone
*                          other than the TSO/E OMVS command.
*           --  argv[8]  = debug level
*
*                          controls whether or not (hidden) debug
*                          messages are sent to the TTY subsidiary file
*                          descriptor or STDERR (after it has been
*                          set up).  These messages contain debug
*                          information, but are backspaced over and
*                          overwritten with blanks, so they would not
*                          usually appear on the screen.  They will
*                          appear in traces, etc.  This option is
*                          meant to work in conjunction with TSO/E
*                          OMVS command debug mode.
*
*                          0 = don't do any debug recording
*                          1 = don't do any debug recording
*                          2 = don't do any debug recording
*                          3 = don't do any debug recording
*
*                          4 = do debug recording, with overwriting
*                              to hide message on display screen
*
*                          5 = do debug recording, but don't try to
*                              overwrite the debug text on the screen
*
*                          6 = do debug recording to syslog
*           --  argv[9]  = screen width for debug messages
*
*                          This value should be set to the width of
*                          the display screen, if the debug level is
*                          set to 4.  It is used when backspacing and
*                          erasing the debug messages.
*
*                          This value may be set to 0, if the debug
*                          level is not 4.
*
*
*           31  argv[10] = remote hostname (or null) -- only 15
*                          bytes of this will fit into the utmpx
*                          entry
*          255  argv[11] = text for TERM environment variable
*
*                          TERM is not set, if this is an empty
*                          string ("").
*
```

```
*
*           15  argv[12] = text for ROWS environment variable
*
*                          ROWS is not set, if this is an empty
*                          string ("").
*
*
*           15  argv[13] = text for COLUMNS environment variable
*
*                          COLUMNS is not set, if this is an empty
*                          string ("").
*           47  argv[14] = path name for SETUID utmpx recording
*                          routine
*
*                          Empty string ("") means use the default
*                          path, which is "/bin/fomtlinc"
*
*
*           15  argv[15] = program name for SETUID utmpx recording
*                          routine
*
*                          Empty string ("") means use the default
*                          program, which is "fomtlinc"
*
*
*           --  argv[16] = SIGCHLD reset flag
*
*                          1 = SIGCHLD will be reset to the default
*                              handling  (This value should seldom
*                              (if ever) be needed -- the main
*                              purpose in the past was to be sure
*                              that NOCLDSTOP was off.)
*
*                          0 = SIGCHLD handling will not be changed
*
*      Other expected input conditions:
*      -------------------------------
*
*      1) Subsidiary TTY must be open, with no controlling terminal
*         established yet.
*
*
*      2) All signals (except perhaps SIGCHLD) should be in their
*         default handling state, before this routine is called.
*         It is OK for signals to be blocked when this routine is
*         called, however.
*
*      3) Any environment variables other than NLSPATH, LC_SYNTAX,
*         LC_MESSAGES, LC_CTYPE, and LC_COLLATE will be passed through
*         to the invoked shell.  The environment variables named here
*         will be gotten rid of before the shell is called.  They
*         will control message catalog processing before the shell is
*         invoked.
*
*      4) Little validity checking is done on the parameters, which
*         are expected to be correct.  This command is not designed to
*         be run from the shell command line.
*
*
*
* Return Value: Does not return to caller
* ------------
*
*
* Non-returning Exits: Does not return to caller
* -------------------
```

FOMTLOUT module for the logout function

The FOMTLOUT module (`/bin/fomtlout`) performs the logout function.

```
*********************************************************************
*
* Function:
* --------
*
* This function uses the utmpx
* functions to locate and remove the caller's USERID from the utmpx
```

```
* file.  This routine is a SETUID program, so that it can write to the
* utmpx file.
*
* note: This routine removes the caller's session from the utmpx file
*       whenever it is called.  If this routine is called erroneously
*       (when the user is not really logging off) it will go ahead and
*       remove the session from the utmpx file.  This will destroy the
*       integrity of the utmpx file.
* Parameters:
* ----------
*
*   1:IN      argc -- usual main() parameters
*
*   2:IN      argv -- usual main() parameters
*
*     argv[0] = program name ("fomtlinc")
*
*     argv[1] = exit status (as a %d-coded integer value)
*
*     argv[2] = (not used)
*
*     argv[3] = (not used)
*
*     argv[4] = debug level (as a %d-coded integer = 0 to 5)
*
*     others -- test only arguments follow
*
*
*
*   assumed file descriptors on entry:
*
*   0 -- manager TTY (needed for ttyname()
*   1 -- might be a debug fd (debug messages written here)
*   2 -- might be an error fd (for system error messages)
*
*
* Return Value:
* ------------
*
*   1-byte exit status contains 2 bits of syscall ID and 6 bits of
*   compressed errno information for the parent.  This information is
*   used only by the TSO/E OMVS command to put out logoff-oriented
*   error messages to the user's terminal.
* Non-returning Exits: none
* -------------------
*
*
* Main Side Effects:
* -----------------
*
*   The caller's session is removed from the utmpx file
```

Appendix C. Accessibility

Accessible publications for this product are offered through IBM Documentation for z/OS (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see How to Send Feedback to IBM to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This book is intended to help the customer plan for, customize, operate, manage, and maintain a z/OS system with z/OS UNIX System Services (z/OS UNIX).

This book primarily documents intended Programming Interfaces that allow the customer to write programs that use z/OS UNIX.

This book also documents information that is NOT intended to be used as Programming Interfaces of z/OS UNIX. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

NOT Programming Interface Information

End NOT Programming Interface Information

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

Glossary

This glossary includes terms and definitions for z/OS UNIX System Services.

The following cross-references are used in this glossary:

1. *See* refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
2. *See also* refers the reader to a related or contrasting term.

3270 pass-through mode
A mode that lets a program running in a shell environment send and receive a data stream or issue TSO/E commands.

absolute address
An address that, without the need for further evaluation, identifies a storage location or a device.

absolute path name
A string of characters used to refer to an object, starting at the highest level (or root) of the directory hierarchy. The absolute path name must begin with a slash (/), which indicates that the path begins at the root.

absolute value
The numeric value of a number regardless of its algebraic sign (positive or negative).

access
The ability to read, update, or otherwise use a resource. Access to protected resources is usually controlled by system software.

access ACL
An access control list (ACL) that provides protection for a file system object.

access authority
One of a range of possible authority levels that control access to protected resources.

access control
In computer security, the process of ensuring that users can access only those resources of a computer system for which they are authorized.

access control list (ACL)
In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights.

access list entry token (ALET)
A token that serves as an index into an access list.

access method
A technique for moving data between main storage and input/output devices.

access method services (AMS)
A multifunction utility named IDCAMS that is used to manage catalogs, devices, and both VSAM and non-VSAM data sets.

access mode
A form of access permitted for a file.

access permission
A group of designations that determine the users who can access a particular file and how the users can access the file. The access permissions are read, write, and run (execute).

accessible
Pertaining to an object for which a client has a valid designator or handle.

accessor environment element (ACEE)
A control block that contains a description of the current user's security environment, including user ID, current connect group, user attributes, and group authorities. An ACEE is constructed during user identification and verification.

account
An entity that contains a set of parameters that define the application-specific attributes of a user, which include the identity, user profile, and credentials.

ACEE
See accessor environment element.

ACL
See access control list.

address
A unique code or identifier for a register, device, workstation, system, or storage location.

address space (ASID)
The range of addresses available to a computer program or process. Address space can refer to physical storage, virtual storage, or both.

address space identifier (ASID)
A unique, system-assigned identifier for an address space.

Advanced Program-to-Program Communication (APPC)
An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

advisory lock
A type of lock that a process holds on a region of a file preventing any other process from locking the region or an overlapping region.

aggregate
A structured collection of data objects that form a data type.

alert
To cause the user's terminal to give some audible or visual indication that an error or some other event has occurred.

ALET
See access list entry token.

algorithm
A set of well-defined rules for the solution of a problem in a finite number of steps.

alias
An alternative name used instead of a primary name.

alias address
An alternative address for a network interface that can be used in place of the real address.

alias name
A name that is used to represent all or part of a command.

allocate
To assign a resource to a specific task.

alphabetic
Pertaining to the set of letters and symbols, excluding digits, used in a language. This set usually consists of the uppercase and lowercase letters plus special symbols (such as $ and _) allowed by a particular language.

alphabetic character
A letter or other symbol, excluding digits, used in a language. Usually the uppercase and lowercase letters A through Z plus other special symbols (such as $ and _) allowed by a particular language.

alphanumeric character
A lowercase or uppercase letter, number, or special symbol.

American National Standards Institute (ANSI)
A private, nonprofit organization whose membership includes private companies, U.S. government agencies, and professional, technical, trade, labor, and consumer organizations. ANSI coordinates the development of voluntary consensus standards in the U.S.

American Standard Code for Information Interchange (ASCII)
A standard code used for information exchange among data processing systems, data communication systems, and associated equipment. ASCII uses a coded character set consisting of 7-bit coded characters.

AMS
See access method services.

ANSI
See American National Standards Institute.

ANSI control character
A control character as defined by the FORTRAN standards of American National Standards Institute (ANSI) and International Organization for Standardization (ISO). It appears at the beginning of each record.

APAR
See authorized program analysis report.

APF
See authorized program facility.

APF-authorized
Pertaining to a program that is authorized by the authorized program facility (APF) to access restricted functions, such as supervisor calls (SVC) or SVC paths.

API
See application programming interface.

APPC
See Advanced Program-to-Program Communication.

application
One or more computer programs or software components that provide a function in direct support of a specific business process or processes.

application program
A program used to communicate with stations in a network, enabling users to perform application-oriented activities.

application programming interface (API)
An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

archive
To copy programs, data, or files to another storage media, usually for long-term storage or security.

archive library
A facility for grouping application-program object files. The archive library file, when created for application-program object files, has a special symbol table for members that are object files.

array
A number of items stored together, which a user can quickly retrieve by supplying the correct index.

ASCII
See American Standard Code for Information Interchange.

ASID
See address space identifier.

See address space.

assembler
A computer program that converts assembly language instructions into object code.

associativity
The order for grouping operands with an operator (either left-to-right or right-to-left).

attach
In z/OS, to create a task that can execute concurrently with the attaching code.

authorized program analysis report (APAR)
A request for correction of a defect in a supported release of an IBM-supplied program.

authorized program facility (APF)
In a z/OS environment, a facility that permits the identification of programs that are authorized to use restricted functions.

automatic conversion
In Enhanced ASCII, the conversion of text data from EBCDIC to ASCII and from ASCII to EBCDIC. This capability makes it easier to port international applications developed on, or for, ASCII systems to z/OS systems.

automatic variable
A variable allocated on entry to a routine and deallocated on the return.

automount rule
A generic or specific entry in an automount map file.

auxiliary storage
All addressable storage other than main storage. See also memory.

awk
A file processing language that is well suited to data manipulation and retrieval of information from text files.

B
See byte.

background
The conditions under which low-priority, noninteractive programs are run.

background process
A process that does not require operator intervention but can be run by the computer while the workstation is used to do other work.

backslash
The character \. The backslash enables a user to escape the special meaning of a character. That is, typing a backslash before a character tells the system to ignore any special meaning the character might have.

backup
Pertaining to a system, device, file, or facility that can be used in the event of a malfunction or loss of data.

base address
An address that is used as a reference point for resolving symbolic references to locations in storage.

base address register
See base register.

base name
The last element to the right of a full path name.

base register
A general purpose register that the programmer chooses to contain a base address.

basic mode
A central processor mode that does not use logical partitioning.

basic partitioned access method (BPAM)
An access method that can be used to create program libraries in direct access storage for convenient storage and retrieval of programs.

batch
A group of records or data processing jobs brought together for processing or transmission.

Pertaining to a group of jobs to be run on a computer sequentially with the same program with little or no operator action.

batch job
A predefined group of processing actions submitted to the system to be performed with little or no interaction between the user and the system.

batch mode
The condition established so that batch processing can be performed.

batch processing
A method of running a program or a series of programs in which one or more records (a batch) are processed with little or no action from the user or operator.

binary file
A file format that does not consist of a sequence of printable characters (text).

binder
See linkage editor.

block special file
A file that provides a low level block access to an input or output device. See also character special file.

Boolean
Characteristic of an expression or variable that can only have a value of true or false.

BPAM
See basic partitioned access method.

brace
Either of the characters left brace ({) and right brace (}). When an object is enclosed in braces, the left brace immediately precedes the object and the right brace immediately follows it.

bracket
Either of the characters left bracket ([) and right bracket (]).

break condition
In the TTY subsystem, a character framing error in which the data is all zeros.

break statement
A C or C++ control statement that contains the keyword break and a semicolon (;). It is used to end an iterative or a switch statement by exiting from it at any point other than the logical end. Control is passed to the first statement after the iteration or switch statement.

breakpoint
A marked point in a process or programmatic flow that causes that flow to pause when the point is reached, usually to allow debugging or monitoring.

build
To convert a product from source code to a binary or executable software product.

built-in shell command
A command that is implemented as part of a shell program. Certain commands are built into the shell in order to improve the performance of shell scripts or to access the shell's internal data structures and variables.

byte (B)
A string that represents a character and usually consists of eight binary digits that are treated as a unit. A byte is the smallest unit of storage that can be addressed directly.

C library
A system library that contains common C language subroutines for file access, string operations, character operations, memory allocation, and other functions.

C shell
A command line processor for UNIX that provides interactive features such as job control and command history.

C++ library
A system library that contains common C++ language subroutines for file access, memory allocation, and other functions.

call
To start a program or procedure, usually by specifying the entry conditions and transferring control to an entry point.

callable service
A set of documented interfaces between the z/OS operating system and higher level applications that want to access functions specified in the Single UNIX Specification and earlier standards.

cancel
To end a task before it is completed.

canonical mode
See line mode.

carriage control character
A character that is used to specify a write, space, or skip operation. See also control character.

carriage return
A keystroke generally indicating the end of a command line.

catalog
A directory of files and libraries, with reference to their locations.

cataloged procedure
A set of job control language (JCL) statements that has been placed in a library and that is retrievable by name.

CBPDO
See Custom-built Product Delivery Option.

CCSID
See coded character set identifier.

CDS
See couple data set.

CECP
See country extended code page.

cell
A logical grouping of users, computers, data, and other resources that share either a common purpose or a common level of trust.

central storage
Storage that is an integral part of the processor unit. Central storage includes both main storage and the hardware system area. UNIX-experienced users refer to central storage as memory.

character class
A named set of characters sharing an attribute associated with the name of the class. The classes and the characters that they contain are dependent on the value of the LC_CTYPE category in the current locale.

character constant
A constant value whose data attribute is character.

character conversion table
A table that converts one or more characters to alternative characters using hexadecimal encoding for the character sets. The character sets are defined in code pages.

character set
A defined set of characters with no coded representation assumed that can be recognized by a configured hardware or software system. A character set may be defined by alphabet, language, script, or any combination of these items.

character special file
A special interface file that provides access to an input or output device, which uses character I/0 instead of block I/0.

character string
A sequence of consecutive characters that are treated as a unit.

character type
A data type that consists of alphanumeric characters.

checksum
The sum of a group of data that is associated with a group of data and that is used for error detection.

child
A node that is subordinate to another node in a tree structure. Only the root node is not a child.

child process
A process that is created by a parent process and that shares the resources of the parent process to carry out a request.

child resource
A secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource.

child segment
In a database, any segment that is dependent on another segment above it (its parent) in the hierarchy.

choice
An option in a pop-up window or menu used to influence the operation of the system.

CINET
See Common INET.

classification rule
A rule used by the workload management component of z/OS to assign a service class.

client
A software program or computer that requests services from a server.

client process
A process that requests services from a server process.

client/server
Pertaining to the model of interaction in distributed data processing in which a program on one computer sends a request to a program on another computer and awaits a response. The requesting program is called a client; the answering program is called a server.

close
To end processing by ending the connection between the file and a program.

code page
A particular assignment of code points to graphic characters. Within a given code page, a code point can have only one specific meaning. A code page also identifies how undefined code points are handled..

code point
A unique bit pattern that represents a character in a code page.

coded character set identifier (CCSID)
A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation.

collating element
The smallest entity used to determine the logical ordering of strings. A collating element consists of either a single character, or two or more characters collating as a single entity. The value of the LC_COLLATE category in the current locale determines the current set of collating elements.

collating sequence
A specified arrangement used in sequencing.

colony address space
An address space in which a physical file system (PFS) can be initialized. The address space can be viewed as a logical extension to the kernel address space.

column
A vertical arrangement of characters or other expressions. Columns are positioned side by side on a page or display.

command
A statement used to initiate an action or start a service. A command consists of the command name abbreviation, and its parameters and flags if applicable.

command alias
An abbreviation of a long command line or a new name for a command. [OSF]

command history
An automatic listing of previously issued commands.

command interpreter
See command language interpreter

command language interpreter
A program that reads commands and changes them into computer instructions.

command line
The blank line on a display where commands, option numbers, or selections can be entered.

command list
A list of commands and statements designed to perform a specific function for the user.

command mode
A state of a system or device in which the user can enter commands.

command name
The first term in a command, a verb, which specifies the action to be performed and is usually followed by operands.

command processor
A module designed to perform a specific function for the user. Users can write command processors in assembler language or in a high-level language. Command processors are started as commands.

command substitution
In UNIX-based operating systems, a shell feature that makes it possible to use the output from one command as an argument to another command.

command-line argument
A part of a command line, delimited by white space. Arguments are used to specify detailed behavior to a program. They are usually either command line options selecting variations in program operation, or path names of files to be processed.

comment
Explanatory text in a program or file that is not translated by the compiler.

commit
To end a unit of work by releasing locks so that the database changes made by that unit of work can be perceived by other processes. This operation makes the data changes permanent.

Common INET (CINET)
A physical file system layer for the address families AF_INET and AF_INET6 that multiplexes sockets across several other physical file systems or transports.

common VTOC access facility (CVAF)
A set of macros that enables programs to access data in the volume table of contents (VTOC) and the VTOC index.

communication
The process of sending or receiving data between two points of a network.

Communications Server
IBM SecureWay Software that supports (a) the development and use of application programs across two or more connected systems or workstations, (b) multiple concurrent connections that use a wide range of protocols, and (c) several application programming interfaces (APIs) that may be called concurrently and that are designed for client/server and distributed application programs.

compatibility
The ability of a device or system to work with another device or system without modification.

compilation unit
A portion of a computer program sufficiently complete to be compiled correctly.

compile
To translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language.

compiler
A program that translates a source program into an executable program (an object program).

compiler option
A keyword that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages.

component
A part of a structured type or value, such as an array element or a record field.

condition
An expression that can be evaluated as true, false, or unknown. It can be expressed in natural language text, in mathematically formal notation, or in a machine-readable language.

condition code
A code that reflects the result of a previous input/output, arithmetic, or logical operation.

condition variable
An object that allows a thread to suspend execution when it finds an untrue condition, and to resume execution when another thread makes the condition true.

configuration
The machines, devices, and programs that make up a system, subsystem, or network.

configure
To describe the interconnected arrangement of the devices, programs, communications, and optional features installed on a system.

conformance document
A document provided by an implementer (such as IBM) that contains implementation details as described in the current POSIX.1 standard.

connection
In data communication, an association established between entities for conveying information.

constant
A language element that specifies an unchanging value. Constants are classified as string constants or numeric constants.

constant field
A field defined by a display format to contain a value that does not change.

context
The address space for a process, hardware registers, and related kernel data structures.

control block
A storage area used by a program to hold control information.

control character
A character whose occurrence in a particular context initiates, modifies, or stops a control function.

control section (CSECT)
The part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.

control statement
In programming languages, a statement that is used to interrupt the continuous sequential processing of programming statements. Conditional statements such as IF, PAUSE, and STOP are examples of control statements.

controlled program
A RACF function with which an installation can control who can run RACF-controlled programs.

controlling process
A session leader that has control of a terminal.

controlling terminal
The active workstation from which the process group for that process was started. Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session.

conversational
Pertaining to a program or a system that conducts a dialog with a terminal user, alternately receiving and transmitting data.

conversion
The process of changing from one form of representation to another. Changing a code point that is assigned to a character in one code page to its corresponding code point in another code page is an example of conversion.

conversion table
A table that contains a set of characters that can be replaced with alternative characters.

Coordinated Universal Time (UTC)
The international standard of time that is kept by atomic clocks around the world.

copy
To read data from a source, leaving the source data unchanged, and to write the same data elsewhere.

counter
A register or storage location used to accumulate the number of occurrences of an event.

country extended code page (CECP)
An EBCDIC code page that is extended by the addition of code points for characters needed in the language used by a specific country. Using country extended code page (CECP) support, a German panel, for example, can be displayed on a French CECP terminal with all common characters displayed correctly

couple data set (CDS)
A data set that contains information related to a sysplex, its systems, cross-system coupling facility (XCF) groups, and their members.

coupling facility
A special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex.

CRC
See cyclic redundancy check.

cross-system coupling facility (XCF)
A component that provides functions to support cooperation between authorized programs running within a sysplex.

CSECT
See control section.

customization
The process of designing a data processing installation or network to meet the requirements of particular users. Activities can include installing additional products, taking advantage of new software features and functions, and enabling or disabling optional features.

CVAF
See common VTOC access facility.

cyclic redundancy check (CRC)
A redundancy check in which the check key is generated by a cyclic algorithm

daemon
A program that runs unattended to perform continuous or periodic functions, such as network control.

DASD
See direct access storage device.

data area
A memory area that is used by a program to hold information.

data definition (DD)
A program statement that describes the features of, specifies relationships of, or establishes the context of data. A data definition reserves storage and can provide an initial value.

data definition name (ddname)
The name of a data definition (DD) statement that corresponds to a data control block that contains the same name.

data definition statement (DD statement)
A job control statement that is used to define a data set for use by a batch job step, started task or job, or an online user.

Data Facility Storage Management Subsystem
See DFSMS.

data integrity
The condition that exists as long as accidental or intentional destruction, alteration, or loss of data does not occur.

data security
The protection of data against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional.

data set
The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. See also file.

data space
A separate area of addressable storage that contains only data. A data space can hold up to 2 gigabytes of data.

data stream
The commands, control codes, data, or structured fields that are transmitted between an application program and a device such as printer or nonprogrammable display station.

data structure
In Open Source Initiative (OSI), the syntactic structure of symbolic expressions and their storage allocation characteristics.

data type
In programming languages, a descriptor of a set of values together with a set of permitted operations. A data type determines the kind of value that a variable can assume or that a function can return.

database (DB)
A collection of interrelated or independent data items that are stored together to serve one or more applications.

DB
See database.

DBCS
See double-byte character set.

DD
See device driver.

See data definition.

DD statement
See data definition statement.

ddname
See data definition name.

deadlock
Unresolved contention for the use of resources.

deallocate
To release a resource that is assigned to a specific task.

debug
To detect, diagnose, and eliminate errors in programs.

debugger
A tool used to detect and trace errors in computer programs.

decimal
Pertaining to a system of numbers to the base 10. The decimal digits range from 0 through 9.

declaration
In the C and C++ languages, a description that makes an external object or function available to a function or a block statement.

default
Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

default access control list (default ACL)
A template used to generate access control lists (ACLs) for the files within a directory. A default ACL is not used to verify permissions.

default ACL
See default access control list.

default directory
The directory name supplied by the operating system if none is specified.

definition
A declaration that reserves storage and can provide an initial value for a data object or define a function.

descriptor
A small, unsigned integer that a UNIX system uses to identify an object supported by the kernel. Descriptors can represent files, pipes, sockets, and other I/O streams.

device
A piece of equipment. Devices can be workstations, printers, disk drives, tape units, or remote systems.

device driver (DD)
A program that provides an interface between a specific device and the application program that uses the device.

DFSMS (Data Facility Storage Management Subsystem)
An operating environment that helps automate and centralize the management of storage. To manage storage, the storage management subsystem (SMS) provides the storage administrator with control over data class, storage class, management class, storage group, and automatic class selection (ACS) routine definitions.

diagnostic
Pertaining to the detection and isolation of an error.

digit
A symbol that represents one of the nonnegative integers smaller than the radix.

direct access storage device (DASD)
A device that allows storage to be directly accessed, such as a disk drive.

direct data set
A data set that has records in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address relative to the beginning of the data set. See also sequential data set.

directory
A type of file that contains the names and controlling information for objects or other directories.

In a hierarchical file system, a grouping of related files and directories. A directory can contain zero or more entries, which refer to other directories and files.

directory default ACL
A model access control list (ACL) that is inherited by subdirectories that are created within the parent directory.

dirty address space
An address space requiring daemon authority that has had an uncontrolled program loaded into it. A dirty address space cannot perform daemon activities.

discretionary access control
A security mechanism that protects information from unauthorized disclosure or modification through owner-controlled access to files.

distributed computing
A method of computing in which large problems are divided into small tasks that are distributed across a network for simultaneous processing. Individual results are then brought together to form the total solution.

DLL
See dynamic link library.

DNS
See Domain Name System.

Domain Name System (DNS)
The distributed database system that maps domain names to IP addresses.

dot
A symbol (.) that indicates the current directory in a relative path name.

dot dot
A symbol (..) in a relative path name that indicates the parent directory.

double quote
See quotation mark.

double-byte character set (DBCS)
A set of characters in which each character is represented by two bytes. These character sets are commonly used by national languages, such as Japanese and Chinese, that have more symbols than can be represented by a single byte.

double-precision
Pertaining to the use of two computer words to represent a number in accordance with the required precision.

DSECT
See dummy control section.

dub
To make an MVS address space known to z/OS UNIX.

dummy control section (DSECT)
A control section that an assembler can use to format an area of storage without producing any object code.

dump
To record or copy, at a particular instant, data from one storage device onto another storage device to protect the data and debug the program.

dynamic
Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time.

dynamic link library (DLL)
A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a DLL can be shared by several applications simultaneously.

dynamic link pack area (dynamic LPA)
A facility for adding additional modules to the to the link pack area (LPA) after the LPA has been created.

dynamic LPA
See dynamic link pack area.

dynamic storage
An area of storage that is explicitly allocated by a program or procedure while it is running.

EBCDIC
See Extended Binary Coded Decimal Interchange Code.

EBCDIC character
Any one of the symbols included in the EBCDIC set.

editor program
A computer program designed to perform such functions as rearrangement, modification, and deletion of data in accordance with prescribed rules.

effective group ID
The current group ID, but not necessarily the user's own ID. For example, a user logged in under a particular group ID might be able to change to another group ID. The ID to which the user changes then becomes the effective group ID.

element
The smallest unit in a table, array, list, set, or other structure. Examples of an element are a value in a list of values and a data field in an array.

ELPA
See extended link pack area.

empty string
A character array whose first element is a null character.

emulation
The use of software, hardware, or both by one system to imitate another system. The imitating system accepts the same data, runs the same programs, and achieves the same results as the imitated system.

enclave
A transaction that can span multiple dispatchable units (service request blocks and tasks) in one or more address spaces and is reported on and managed as a unit.

end-of-file (EOF)
A code that signals that the last record of a file has been read.

enforced lock
A type of lock that a process holds on a region of a file preventing any other process from accessing that region with read or write system calls. In addition, the create command is prevented from truncating the files.

entry
An element of information in a table, list, queue, or other organized structure of data or control information.

entry point
The address or label of the first instruction processed or entered in a program, routine, or subroutine.There might be a number of different entry points, each corresponding to a different function or purpose.

environment
The settings for shell variables and paths that are set when the user logs in. These variables can be modified later by the user.

environment variable
A variable that defines an aspect of the operating environment for a process. For example, environment variables can define the home directory, the command search path, the terminal in use, or the current time zone.

EOF
See end-of-file.

epoch
The time and date corresponding to 0 in an operating system's clock and time-stamp values. For most versions of the UNIX operating system, the epoch is 00:00:00 GMT, 01 January 1970. System time is measured as the number of seconds past the epoch.

equivalence class
A grouping of characters or character strings that are considered equal for purposes of collation. For example, many languages place an uppercase character in the same equivalence class as its lowercase form, but some languages distinguish between accented and unaccented character forms for the purpose of collation.

error condition
The state that results from an attempt to run instructions in a computer program that are not valid or that operate on data that is not valid.

escape sequence
A character that is preceded by a \ (backslash) and is interpreted to have a special meaning to the operating system.

ESQA
See extended system queue area.

executable file
A file that contains programs or commands that perform operations on actions to be taken.

executable program
A program in a form suitable for execution by a computer. The program can be an application or a shell script.

exit routine
A program that receives control from another program in order to perform specific functions.

export
In Network File System (NFS), to make file systems on a server available to remote clients.

Extended Binary Coded Decimal Interchange Code (EBCDIC)
A coded character set of 256 8-bit characters developed for the representation of textual data.

extended link pack area (ELPA)
The portion of virtual storage above 16MB that contains frequently used modules.

extended system queue area (ESQA)
A major element of z/OS virtual storage above the 16MB line. This storage area contains tables and queues relating to the entire system. It duplicates above the 16MB line the system queue area (SQA) .

extent
A continuous space on a disk, direct-access storage volume, or diskette that is occupied by or reserved for a particular data set, data space, or file.

external
In programming languages, pertaining to a language object that has a scope that extends beyond one module, for example, the entry names of a module.

external link
A symbolic link that contains the name of an object that is outside the hierarchical file system.

Extra Performance Linkage (XPLINK)
A type of call linkage that can improve performance in an environment of frequent calls between small functions.

Federal Information Processing Standard (FIPS)
A standard produced by the National Institute of Standards and Technology when national and international standards are nonexistent or inadequate to satisfy the U.S. government requirements.

FIFO special file
A type of file with the property that data written to such a file is read on a first-in-first-out (FIFO) basis.

file
A collection of related data that is stored and retrieved by an assigned name. See also data set.

file access permission
A designation that determines who can access a particular file and how the user can access the file.

file default ACL
A model access control list (ACL) that is inherited by files that are created within the parent directory.

file descriptor
A positive integer or a data structure that uniquely identifies an open file for the purpose of file access.

file lock
A means to limit or deny access to a file by other users. A file lock can be a read lock or a write lock.

file mode
An object containing the file permission bits and other characteristics of a file.

file mode creation mask
A pattern of characters that is used to establish maximum permissions that can then be applied to individual access control list (ACL) entries.

file name
A name assigned to identify a file.

file offset
The byte position in the file where the next I/O operation begins.

file owner
The user who has the highest level of access authority to a file, as defined by the file.

file permission bit
Information about a file that is used, along with other information, to determine whether a process has read, write, or execute permission to a file. The use of file permission bits is described in file access permissions.

file pointer
An identifier that indicates a structure containing the file name.

file system
A collection of files and certain attributes associated with those files.

file system owner
The system that coordinates sysplex activity for a particular file system.

file tag
A file attribute that identifies the character set of the text data within a file and indicates whether the file is eligible for automatic conversion.

File Transfer Protocol (FTP)
In TCP/IP, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

filter
A command that reads standard input data, modifies the data, and sends it to standard output. A pipeline usually has several filters.

FIPS
See Federal Information Processing Standard.

fixed-length record
A record whose length is established as an attribute of the file in which it is stored, and cannot be changed. Every record in such a file has the same length, which is specified by the record length attribute (LRECL) of the file.

flag
An indicator or parameter that shows the setting of a switch.

floating-point number
A real number represented by a pair of distinct numerals. The real number is the product of the fractional part, one of the numerals, and a value obtained by raising the implicit floating-point base to a power indicated by the second numeral.

floating-point register (FPR)
A register used to manipulate data in a floating-point representation system. [I][A]

fold
To translate the lowercase characters of a character string into uppercase.

foreground
In multiprogramming, the environment in which high-priority programs are run.

foreground process group
A group whose member processes have privileges that are denied to background processes when the controlling terminal is being accessed. Each controlling terminal can have only one foreground process group.

foreign cell
A cell other than the one to which the local machine belongs. A foreign cell and its binding information are stored in either Global Directory Service (GDS) or the Domain Name System (DNS).

fork
To create and start a child process.

forked address space
An address space created by a fork function. A forked address space is perceived by MVS to be a batch job.

formatted file
A file that is arranged with particular characteristics, such as line spacing, headings, and number of characters and lines per page.

FPR
See floating-point register.

free space
The total amount of unused space in a page, data set, file, or storage medium. Free space is the space that is not used to store records or control information.

FRR
See functional recovery routine.

FTP
See File Transfer Protocol.

fully qualified name
A qualified name that includes all names in the hierarchical sequence above the structure member to which the name refers, as well as the name of the member itself.

function
Any instruction or set of related instructions that perform a specific operation.

function call
An expression that transfers the path of execution from the current function to a specified function (the called function). A function call contains the name of the function to which control is transferred and a parenthesized list of values

function key
A keyboard key that can be programmed to perform certain actions.

functional recovery routine (FRR)
A z/OS recovery and termination manager that enables a recovery routine to gain control in the event of a program interrupt.

function shipping
The process of requesting function from the owning file system and returning the response to the requester through XCF communications.

G11N
See globalization.

GDG
See generation data group.

GDS
See generation data group.

general purpose register (GPR)
An explicitly addressable register that can be used for a variety of purposes (for example, as an accumulator or an index register).

generation data group (GDG)
A chronological collection of historically related data sets that do not use the Virtual Storage Access Method (VSAM); each data set is called a generation data set.

generation data set (GDS)
One of the data sets in a generation data group (GDG); a GDS is historically related to the other data sets in the group.

Generic Security Services API
See Generic Security Services application programming interface.

Generic Security Services application programming interface (Generic Security Services API, GSS API)
A common application programming interface (API) for accessing security services.

GID
See group ID.

globalization (G11N)
In computing, the provision of a single software solution that has (1) multicultural support and (2) a user interface and documentation that is available in one or more languages.

goal mode
A mode of processing in which the active service policy determines system resource management.

GPR
See general purpose register.

graphic character
A visual representation of a character, other than a control character, that is normally produced by writing, printing, or displaying.

group
A collection of users who can share access authorities for protected resources.

group ID (GID)
In the UNIX operating system, an integer that uniquely identifies each group of users to the operating system.

group name
A name that uniquely identifies a group of users to the system. The group name contains 1 - 8 alphanumeric characters, beginning with an alphabetic character or one of these special characters: #, $, or >.

GSS API
See Generic Security Services application programming interface.

H/W
See hardware.

halfword
A contiguous sequence of bits or characters that constitutes half a computer word and can be addressed as a unit.

handler
A software routine that controls a program's reaction to specific external events, such as an interrupt handler.

hardware (H/W)
The physical components of a computer system.

header
System-defined control information that precedes user data.

header file
See include file.

hierarchical storage management (HSM)
A function that automatically distributes and manages data on disk, tape, or both by regarding devices of these types and potentially others as levels in a storage hierarchy that range from fast, expensive devices to slower, cheaper, and possibly removable devices. The objectives are to minimize access time to data and maximize available media capacity.

High Level Assembler
An IBM licensed program that translates symbolic assembler language into binary machine language.

high-level language (HLL)
A programming language that provides some level of abstraction from assembler language and independence from a particular type of machine.

high-order
The most significant; leftmost. For example, bit 0 in a register is the high-order bit.

Hiragana
One of the two common Japanese phonetic alphabets (the other is katakana). The symbols are cursive or curvilinear in style. Hiragana syllables are typically used in the representation of native Japanese words and grammatical particles.

history file
A file in which a record is kept of shell commands that are executed.

HLL
See high-level language.

HSM
See hierarchical storage management.

Huffman coding
A character-coding technique to compress data.

i-node
The internal structure that describes the individual files in the UNIX file system. An i-node contains the node, type, owner, and location of a file.

i-node number
A number specifying a particular i-node file in the file system.

I/O
See input/output.

IAR
See instruction address register.

ID
See identifier.

identifier (ID)
A sequence of bits or characters that identifies a user, program, device, or system to another user, program, device, or system.

IEEE
See Institute of Electrical and Electronics Engineers.

if statement
A conditional statement that specifies a condition to be tested and the action to be taken if the condition is satisfied.

include file
A text file that contains declarations that are used by a group of functions, programs, or users.

index
A table that contains key values or references for locating information in an indexed file.

informational message
A message that provides information about the system and is not the result of an error condition. This message does not require a response.

inherit
To copy resources or attributes from a parent to a child.

initial program load
The process of loading the operating system and other basic software into main storage.

input
Data entered for processing or storage.

input redirection
The specification of an input source other than the standard one.

input stream
A sequence of control statements and data submitted to an operating system by an input device.

input/output (I/O)
Pertaining to a device, process, channel, or communication path involved in data input, data output, or both.

installation
A particular computing system, including the work it does and the people who manage it, operate it, apply it to problems, service it, and use the results it produces.

Institute of Electrical and Electronics Engineers (IEEE)
A professional society accredited by the American National Standards Institute (ANSI) to issue standards for the electronics industry.

instruction
A program statement that specifies an operation to be performed by the computer, along with the values or locations of operands. This statement represents the programmer's request to the processor to perform a specific operation. [OSF]

instruction address register (IAR)
A register in the processor that contains the address of the next instruction to be processed.

integer
A positive or negative whole number, or zero.

integer expression
An arithmetic expression with only integer type values.

Interactive Problem Control System (IPCS)
A component of MVS and z/OS that permits online problem management, interactive problem diagnosis, online debugging for disk-resident abend dumps, problem tracking, and problem reporting.

interactive processing
A processing method in which each operator action causes a response from the program or the system.

Interactive System Productivity Facility (ISPF)
An IBM licensed program that serves as a full-screen editor and dialog manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogs between the application programmer and terminal user.

interface
A shared boundary between independent systems. An interface can be a hardware component used to link two devices, a convention that supports communication between software systems, or a method for a user to communicate with the operating system, such as a keyboard.

International Organization for Standardization (ISO)
An international body charged with creating standards to facilitate the exchange of goods and services as well as cooperation in intellectual, scientific, technological, and economic activity.

Internet
The worldwide collection of interconnected networks that use the Internet suite of protocols and permit public access.

Internet Protocol (IP)
A protocol that routes data through a network or interconnected networks. This protocol acts as an intermediary between the higher protocol layers and the physical network.

interoperability
The ability of a computer or program to work with other computers or programs.

interprocess communication (IPC)
The process by which programs send messages to each other. Sockets, semaphores, signals, and internal message queues are common methods of interprocess communication.

interrupt
Suspension of a process, such as execution of a computer program, caused by an external event and performed in such a way that the process can be resumed.

IP
See Internet Protocol.

IP socket
The port that is concatenated with the Internet Protocol (IP) address.

IPC
See interprocess communication.

IPCS
See Interactive Problem Control System.

ISO
See International Organization for Standardization.

ISPF
See Interactive System Productivity Facility.

item
The data in one line of an indexed field.

JCL
See job control language.

JES
See Job Entry Subsystem.

JES2
An MVS subsystem that receives jobs into the system, converts them to internal format, selects them for execution, processes their output, and purges them from the system. In an installation with more than one processor, each JES2 processor independently controls its job input, scheduling, and output processing.

JES3
An MVS subsystem that receives jobs into the system, converts them to internal format, selects them for execution, processes their output, and purges them from the system. In complexes that have several loosely coupled processing units, the JES3 program manages processors so that the global

processor exercises centralized control over the local processors and distributes jobs to them using a common job queue.

job control language (JCL)
A command language that identifies a job to an operating system and describes the job's requirements.

Job Entry Subsystem (JES)
An IBM licensed program that receives jobs into the system and processes all output data that is produced by those jobs.

job name
The name of the job as identified to the system. For an interactive job, the job is assigned the name of the workstation at which the job was started; for a batch job, the name is specified in the command used to submit the job.

job step
The execution of a computer program explicitly identified by a job control statement. A job may specify that several job steps be executed. [A]

jump
In the running of a computer program, a departure from the implicit or declared order in which instructions are being run.

justify
To align text so that the margins are even

Kanji
A graphic character set consisting of symbols used in Japanese ideographic alphabets. Each character is represented by 2 bytes.

Katakana
A Japanese phonetic syllabary used primarily for foreign names and place names and words of foreign origin.

Kerberos
A network authentication protocol that is based on symmetric key cryptography. Kerberos assigns a unique key, called a ticket, to each user who logs on to the network. The ticket is embedded in messages that are sent over the network. The receiver of a message uses the ticket to authenticate the sender.

kernel
The part of an operating system that contains programs for such tasks as input/output, management and control of hardware, and the scheduling of user tasks.

kernel address space
The address space containing the MVS support for z/OS UNIX services. This address space can also be called the kernel.

keyboard
An input device consisting of various keys that allows the user to input data, control cursor and pointer locations, and control the dialog with the workstation.

keyword
One of the predefined words of a programming language, artificial language, application, or command.

keyword parameter
A parameter that consists of a keyword followed by one or more values.

kill character
A character that deletes a line of characters entered after a prompt.

Korn shell
A command interpreter developed for UNIX, which forms the basis for the z/OS shell.

label
One or more characters used to identify a statement or an item of data in a computer program.

labeled statement
A programming language statement that contains one or more identifiers followed by a colon and a statement.

last-in first-out (LIFO)
A queuing technique in which the next item to be retrieved is the item most recently placed on the queue.

Latin 1
See Latin alphabet no. 1.

Latin alphabet
An alphabet composed of the letters a - z and A - Z with or without accents and ligatures.

Latin alphabet no. 1 (Latin 1, Latin-1)
The 190 characters used in most of Western Europe, North America, Central and South America . There are other Latin alphabets such as Latin-2 and Latin-3 that correspond to some of the other ISO/IEC 8859 character sets. The numbering scheme is neither rational nor orderly.

Latin-1
See Latin alphabet no. 1.

Lempel-Ziv (LZ)
A technique for compressing data. This technique replaces some character strings, which occur repeatedly within the data, with codes. The encoded character strings are then kept in a common dictionary, which is created as the data is being sent.

level
In a database, the successive vertical dependencies in a hierarchical structure.

lexical analyzer
A program that analyzes input and breaks it into categories, such as numbers, letters, or operators.

library
A collection of model elements, including business items, processes, tasks, resources, and organizations.

library lookaside (LLA)
A z/OS facility that reduces library I/O activity by keeping selected directory entries and modules in storage, instead of making repetitive searches of DASD

licensed program (LP)
A separately priced program and its associated materials that have a copyright and are offered to customers under the terms and conditions of a licensing agreement.

LIFO
See last-in first-out.

line
On a terminal, one or more characters entered before a return to the first printing or display position, or accepted by the system as a single block of output.

line editor
An editor that displays data one line at a time and that allows data to be accessed and modified only by entering commands.

line mode
An input-processing mode in which input is collected and processed one line at a time.

link
In a file system, a connection between an i-node and one or more file names associated with it.

link count
The number of directory entries that refer to a particular file. [POSIX.1]

link list
The list of libraries searched by the control program (after the job pack, task library, step library, job library, and link pack area have been searched) for any load that does not provide a specific data control block to be used. In MVS, the system name is LNKLST.

link pack area (LPA)
The portion of virtual storage below 16MB that contains frequently used modules.

link-edit
To create a loadable computer program by means of a linkage editor.

linkage editor
A computer program for creating load modules from one or more object modules or load modules by resolving cross-references among the modules and, if necessary, adjusting addresses.

literal
A symbol or a quantity in a source program that is itself data, rather than a reference to data.

LLA
See library lookaside.

load
To bring all or part of a computer program into memory from auxiliary storage so that the computer can run the program.

load module
A program in a form suitable for loading into main storage for execution.

loader
A program that copies an executable file into main storage so that the file can be run.

local
Pertaining to a device, file, or system that is accessed directly from a user's system, without the use of a communication line.

locale
A setting that identifies language or geography and determines formatting conventions such as collation, case conversion, character classification, the language of messages, date and time representation, and numeric representation.

lock
A mechanism with which a resource is restricted for use by the holder of the lock.

log in
To connect to a computer system or network by entering identification and authentication information at the workstation.

logger
A functional unit that records events and physical conditions, usually with respect to time.

logical mount
A mount that attaches a file system to the root directory or to a directory of another file system so that the files and directories on the file system can be referenced. The attached file system can consist of a file or many files and directories.

logical operator
A symbol, such as AND, OR, or NOT, that represents an operation on logical expressions.

logical record
A group of logically related fields. Portions of the same logical record may be located in different physical records, and several logical records or parts of several logical records may be located in one physical record.

logically partitioned mode
A capability provided by the Processor Resource/System Manager (PR/SM) that allows a single processor to run multiple operating systems using separate sets of system resources, or logical partitions.

login name
A string of characters that uniquely identifies a user to the system.

logon
The process of connecting to a computer system, network, or terminal session

loop
A sequence of instructions performed repeatedly.

low-order
The least significant, or rightmost, example. For example, in a 32-bit register (0 through 31), bit 31 is the low-order bit.

LP
See licensed program.

LPA
See link pack area.

LZ
See Lempel-Ziv.

machine instruction
A binary number that directs the operation of a processor. Compilers and assemblers convert source instructions to machine instructions.

magic number
A numeric or string constant in a file that indicates the file type.

main function
A function that has the identifier main. Each program must have exactly one function named main. The main function is the first user function that receives control when a program starts to run.

main program
The first program unit to receive control when a program is run.

main storage
The part of internal storage into which instructions and other data must be loaded for running or processing.

mainline routine
The first subroutine encountered when link-editing.

master address space
The virtual storage used by the master scheduler task.

MBCS
See multibyte character set.

medium
The material on which computer information is stored. Examples of media are diskettes, CDs, and tape.

member
A data object in a structure, a union, or a library.

memory
Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent running or processing .See also auxiliary storage.

Message Passing Interface (MPI)
A library specification for message passing. MPI is a standard application programming interface (API) that can be used with parallel applications and that uses the best features of a number of existing message-passing systems.

message queue
A set of messages that are waiting to be processed by a program or to be sent to a terminal, display, or workstation.

metacharacter
In UNIX, a character that has special meaning to the shell.

migrate
To install a new version or release of a program to replace an earlier version or release.

MIME
See Multipurpose Internet Mail Extensions.

model ACL
See default access control list.

modem (modulator-demodulator)
A device that converts digital data from a computer to an analog signal that can be transmitted on a telecommunication line, and converts the analog signal received to data for the computer.

modulator-demodulator
See modem.

module
A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

mount
To make a file system accessible.

mount point
In Linux operating systems and in UNIX operating systems such as AIX®, the directory at which a file system is mounted and under which other file systems may be mounted.

MPI
See Message Passing Interface.

multibyte character set (MBCS)
A character set that represents single characters with more than a single byte.

multilevel security
A security policy that allows the classification of data and users based on a system of hierarchical security levels combined with a system of non-hierarchical security categories. The system imposes mandatory access controls restricting which users can access data based on a comparison of the classification of the users and the data.

Multiple Virtual Storage (MVS)
An IBM operating system that accesses multiple address spaces in virtual storage.

multiprocessing
Simultaneous processing by multiple central-processing units.

Multipurpose Internet Mail Extensions (MIME)
An Internet standard that allows different forms of data, including video, audio, or binary data, to be attached to e-mail without requiring translation into ASCII text.

mutual exclusion lock
A lock that excludes all threads other than the lock holder from any access to the locked resource.

MVS
See Multiple Virtual Storage.

named pipe
A pipe that an application opens by name in order to write data into or read data from the pipe. Using a named pipe facilitates communication between a sending process and a receiving process.

namespace
A construct that provides the appearance of isolation for a specific system resource. to a process within a namespace, only resources local to that specific namespace is available.

network
In data communication, a configuration in which two or more locations are physically connected for the purpose of exchanging data.

Network File System (NFS)
A protocol, developed by Sun Microsystems, Incorporated, that allows a computer to access files over a network as if they were on its local disks.

newline character (NL)
A control character that causes the print or display position to move down one line.

NFS
See Network File System.

NL
See newline character.

node
In communications, an end point of a communication link or a junction common to two or more links in a network. Nodes can be processors, communication controllers, cluster controllers, terminals, or workstations. Nodes can vary in routing and other functional capabilities.

NUL
See null character.

NULL
In the C and C++ languages, a pointer that does not point to a data object.

null character (NUL)
A control character with the value of X'00' that represents the absence of a displayed or printed character.

null string
A character or bit string with a length of zero.

null value
A parameter position for which no value is specified.

null wide-character code
A wide-character code with all bits set to zero.

null-terminated
Pertaining to a character string that ends with a zero byte.

numeric
Pertaining to any of the digits 0 through 9.

numeric constant
A constant that expresses an integer, a real number, or a complex number.

object code
Machine-executable instructions, usually generated by a compiler from source code written in a higher level language. Object code might itself be executable or it might require linking with other object code files.

object file
A member file in an object library.

object library
An area on a direct access storage device used to store object programs and routines.

object module
A set of instructions in machine language that is produced by a compiler or assembler from a subroutine or source module and can be input to the linking program. The object module consists of object code.

object program
A fully compiled or assembled program that is ready to be loaded into the computer. An object program consists of object modules.

OMVS
The portion of a RACF profile that contains information about users of z/OS UNIX System Services, such as attributes.

OMVS segment
The portion of a RACF profile that contains logon information for z/OS UNIX users and groups.

open file
A file that is currently associated with a file descriptor.

open system
A system that complies with industry-defined interoperability standards. An open system can be connected to other systems complying with the same standards.

operand
An argument to a command that is generally used as an object supplying information to a utility necessary to complete its processing. Operands generally follow the options in a command line.

operating system (OS)
A collection of system programs that control the overall operation of a computer system.

operation
A specific action (such as add, multiply, or shift) that the computer performs when requested.

optimize
To improve the speed of a program or to reduce the use of storage during processing.

option
A specification in a statement that can influence the running of the statement.

OS
See operating system.

output
The result of processing data. Output can be displayed, printed, stored, or passed to another process.

output file
A database or device file that is opened with the option to allow records to be written.

output list
A list of variables from which values are written to a file or device.

output redirection
The specification of an output destination other than the standard one.

output stream
Messages and other output data that an operating system or a processing program displays on output devices.

overflow exception
A condition caused by the result of an arithmetic operation having a magnitude that exceeds the largest possible number.

owner
In UNIX-based operating systems, the user name associated with a file. The owner and the superuser control access to the file.

padding
Bytes inserted in the data stream to maintain alignment of the protocol requests on natural boundaries. Padding increases the ease of portability to some machine architectures.

page
A fixed-length block of instructions, data, or both instructions and data that can be transferred between active physical memory and external page storage.

page frame
In real storage, a storage location having the size of a page.

parent directory
The directory one level above the current directory. An object's parent directory is the directory that contains the names and controlling information for the object. If the object is named in more than one directory, it has multiple parent directories.

parent process
A process that is created to carry out a request or set of requests. The parent process, in turn, can create child processes to process requests for the parent.

parent process ID (PPID)
An attribute of a new process identifying the parent of the process. The parent process ID of a process is the process ID of its creator for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-dependent system process.

parent segment
In a database, a segment that has one or more dependent segments (its children) hierarchically below it.

parity
The state of being either even-numbered or odd-numbered.

parity check
A test to determine whether the number of ones or zeros in an array of binary digits is odd or even.

parse
To break down a string of information, such as a command or file, into its constituent parts.

parser
A program that interprets user input and determines what to do with the input.

partitioned data set (PDS)
A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. See also sequential data set.

partitioned data set extended (PDSE)
A data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets (PDSs).

password phrase
A string consisting of mixed-case letters, numbers, and special characters, including blanks, that is used to control access to data and systems.

path
In a network environment, the route between any two nodes.

path name
A name that specifies all directories leading to a file plus the file name itself.

pattern matching
The specification of a pattern of characters for search purposes.

PDS
See partitioned data set.

PDSE
See partitioned data set extended.

performance
A measure of a system's ability to perform its functions, including response time, throughput, and number of transactions per second.

period
The symbol ".". The term dot is used for the same symbol when referring to a Web address or file extension. This character is named <period> in the portable character set.

permanent storage
A storage device whose contents cannot be modified.

permission
The ability to access a protected object, such as a file or directory. The number and meaning of permissions for an object are defined by the access control list.

PFS
See physical file system.

PGID
See process group ID.

phase
A distinct part of a process in which related operations are performed.

physical file
A database file that describes how data is to be presented or received from a program and how data is actually stored in the database. A physical file contains one record format and one or more members.

physical file system (PFS)
The part of the operating system that handles the actual storage and manipulation of data on a storage medium.

physical unit (PU)
In SNA, one of three types of network addressable units (NAUs). A PU exists in each node of an SNA network to manage and monitor, at the request of a system services control point logical unit (SSCP-LU) session, the resources (such as attached links and adjacent link stations) of a node.

PID
See process ID.

pipe
An interprocess communication mechanism that connects an output file descriptor to an input file descriptor. Usually the standard output of one process is connected to the standard input of another, forming a pipeline.

pipeline
A direct, one-way connection between two or more processes.

pointer
A data element or variable that holds the address of a data object or a function.

polling
Interrogation of devices for such purposes as avoiding contention, determining operational status, or determining readiness to send or receive data.

port
An end point for communication between applications, generally referring to a logical connection. A port provides queues for sending and receiving data. Each port has a port number for identification.

To modify a computer program that runs on a given system to enable it to run on a different system.

port number
The part of a socket address that identifies a port within a host.

portability
The ability of a program to run on more than one type of computer system without modification.

portable file name character set
The set of characters from which portable file names must be constructed to be portable across implementations conforming to the ISO POSIX-1 standard and to ISO/IEC 9945.

Portable Operating System Interface (POSIX)
An IEEE family of standards designed to provide portability between operating systems that are based on UNIX. POSIX describes a wide spectrum of operating-system components ranging from C language and shell interfaces to system administration

Portable Operating System Interface for Computer Environments
See Portable Operating System Interface.

positional parameter
A variable within a shell program. Positional parameters are assigned from the shell's arguments when the shell is invoked

POSIX
See Portable Operating System Interface.

POSIX open system environment (POSIX OSE)
The open system environment in which the standards included are not in conflict with ISO/IEC and consist of: International Standards and Profiles, developed by ISO, IEC, or CCITT; Regional Standards and Profiles, developed by a group recognized as an official body by a regional governmental entity, such as the European Community; and National Information Technology Standards and Profiles, developed by a national standards body recognized as such by ISO, IEC, or CCITT, as appropriate.

POSIX OSE
See POSIX open system environment.

PPID
See parent process ID.

PPTP
See protocol.

precedence
The priority system for grouping different types of operators with their operands.

precision
A measure of the ability to distinguish between nearly equal values.

predefined macro
In C/C++, an identifier predefined by the compiler, which will be expanded by the preprocessor during compilation.

preprocessor
A routine that performs initial processing and translation of source code or data prior to compiling the source code or processing the data in another program such as an emulator.

print file
A file that is created for the purpose of printing data. A print file includes information to be printed and, optionally, some of the data.

privileged user
A user logged into an account with root user authority.

procedure
A sequenced set of statements that may be used at one or more points in one or more computer programs, and that usually has one or more input parameters and yields one or more output parameters. [T]

process
An instance of a program running on a system and the resources that it uses.

process accounting
An analysis of the way that each process uses the processing unit, memory, and I/O resources.

process group
A collection of processes in a system that is identified by a process group ID.

process group ID (PGID)
The unique identifier representing a process group during its lifetime. A process group ID is a positive integer that is not reused by the system until the process group lifetime ends.

process ID (PID)
The unique identifier that represents a process. A process ID is a positive integer and is not reused until the process lifetime ends. Every process will have a unique process ID within its local PID namespace as well as within any ancestor PID namespaces.

processor
In a computer, the part that interprets and executes instructions. Two typical components of a processor are a control unit and an arithmetic logic unit.

production system
A system on which application programs that are already developed and tested run on a regular basis.

profile
A file containing customized settings for a system or user.

program
A prepared sequence of instructions to the system to accomplish a defined task. In POSIX.2, a program encompasses applications written in the shell command language, complex utility input languages, and high-level languages (HLLs).

program CCSID
In Enhanced ASCII, a 16-bit value that identifies the current character set of text strings within a program.

program check
A condition that occurs when programming errors are detected by a processor during execution.

program control
An RACF function with which an installation can control who runs RACF-controlled programs.

program counter
See instruction address register.

program status word (PSW)
An area in storage used to indicate the order in which instructions are executed, and to hold and indicate the status of the computer system.

prolog
A user-written definition of an application program, record, or table. A prolog is used for documentation.

prompt
A message or a displayed symbol that requests information or user action. The user must respond to allow the program to proceed.

protocol (PPTP)
A set of rules controlling the communication and transfer of data between two or more devices or systems in a communication network.

PSW
See program status word.

PU
See physical unit.

qualified name
A data name explicitly accompanied by a specification of the class to which it belongs in a specified classification system.

qualifier
A modifier that makes a name unique.

query
In interactive systems, an operation at a workstation that elicits a response from the system.

queue
A data structure for processing work in which the first element added to the queue is the first element processed. This order is referred to as first-in first-out (FIFO).

quiesce
To end a process or shut down a system after allowing normal completion of active operations.

quotation mark
The characters " and '.

quote
To mask the special meaning of certain characters, causing the characters to be taken literally.

RACF
See Resource Access Control Facility.

RACF-indicated
Pertaining to a data set for which the RACF indicator is set on. If a data set is RACF-indicated, a user can access the data set only if a RACF profile or an entry in the global access checking table exists for that data set.

RACF-protected
Pertaining to resources that are defined to RACF. A data set that is RACF-protected by a discrete profile must also be RACF-indicated.

RE
See regular expression.

read lock
A lock that prevents any other process from setting a write lock on any part of the protected area.

real GID
See real group ID.

real group ID (real GID)
For each user, the group ID defined in the password file.

real storage
The main storage in a virtual storage system. Physically, real storage and main storage are identical. Conceptually, however, real storage represents only part of the range of addresses available to the user of a virtual storage system.

real UID
See real user ID.

real user ID (real UID)
For each user, the user ID that is specified in the /etc/passwd file.

reason code
A value used to indicate the specific reason for an event or condition.

record
In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures.

record name
A user-defined name for a record. The name is listed in a record description entry.

recovery procedure
An action performed by the operator when an error message appears on the display screen. This action usually permits the program to continue or permits the operator to run the next job.

recursion
A programming technique in which a program or routine calls itself to perform successive steps in an operation, with each step using the output of the preceding step.

redirect
To divert data from a process to a file or device to which it would not normally go.

redirection
In a shell, a method of associating files with the input or output of commands.

reentrant
The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

region
A contiguous area of virtual storage that has common characteristics and that can be shared between processes.

register
An internal computer component capable of storing a specified amount of data and accepting or transferring this data rapidly.

regular expression
A mechanism for selecting specific strings from a set of character strings.

A sequence of characters or symbols constructed according to the rules defined in POSIX.2 2.8.

regular file
A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. [POSIX.1]

relative path name
A string of characters that is used to refer to an object and that starts at some point in the directory hierarchy other than the root. The starting point is frequently a user's current directory.

relative record number (RRN)
A number that expresses the location of a record in relation to a base position in the file containing it.

remote
Pertaining to a system, program, or device that is accessed through a communication line.

remote terminal
A terminal attached to a system through a data link.

reset
To cause a counter to take the state corresponding to a specified initial number.

resource
A facility of a computing system or operating system required by a job, task, or running program. Resources include main storage, input/output devices, the processing unit, data sets, files, libraries, folders, application servers, and control or processing programs.

Resource Access Control Facility (RACF)
An IBM licensed program that provides access control by identifying users to the system; verifying users of the system; authorizing access to protected resources; logging unauthorized attempts to enter the system; and logging accesses to protected resources.

restore
To return to an original value or image, for example, to restore data to main storage from auxiliary storage.

restricted shell
A facility that provides controlled, limited access to specified users.

resume
To continue execution of an application after an activity has been suspended.

retrieve
To locate data in storage and read it so that it can be processed, printed, or displayed.

return code
A value returned by a program to indicate the result of its processing. Completion codes and reason codes are examples of return codes.

return statement
A control statement in a programming language that contains the word "return" followed by an optional expression and a semicolon.

return value
See return code.

rollback
The process of restoring data that was changed by an application program or user.

root
The UNIX definition for a directory that is the base for all other directories.

root directory
The directory that contains all other directories in the system.

root file system
The basic file system onto which all other file systems can be mounted. The root file system contains the operating system files that run the rest of the system.

root user
A system user who operates without restrictions. A root user has the special rights and privileges needed to perform administrative tasks.

routine
A program or sequence of instructions called by a program. Typically, a routine has a general purpose and is frequently used.

row
A horizontal arrangement of characters or other expressions.

RRN
See relative record number.

run
To cause a program, utility, or other machine function to be performed.

runtime library
A library that is loaded dynamically and used during execution time.

SA
See system administrator.

SAF
See System Authorization Facility.

SBCS
See single-byte character set.

Scalable Parallel 2 (SP2)
IBM's parallel UNIX system: effectively parallel AIX systems on a high-speed network.

scheduler
A computer program that performs functions such as scheduling, initiation, and termination of jobs.

SDSF
See System Display and Search Facility.

SDUMP
See system dump.

search path
A list of directories searched by the shell when a command path name is not specified.

security
The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

security administrator
A programmer who manages, protects, and controls access to sensitive information.

segment
A part of a program that can be run without the entire program being in main storage.

semaphore
An indicator used to control access to a file. For example, in a multiuser application, a semaphore is a flag that prevents simultaneous access to a file.

separator
A punctuation character that separates parts of a command or file, or that delimits character strings.

sequential data set
A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape.

Serial Line Internet Protocol (SLIP)
An Internet protocol that connects a computer to the Internet using a serial line.

serialization
The consecutive ordering of items.

server
A software program or a computer that provides services to other software programs or other computers.

server process
A process that provides services to client processes.

service class
A group of work that has the same service goals or performance objectives, resource requirements, or availability requirements. For workload management, a service goal and, optionally, a resource group is assigned to a service class.

service request block (SRB)
A control block that represents a routine that performs a particular function or service in a specified address space.

session leader
A process that has created a session.

shared address space
A type of address space shared by multiple UNIX System Services (z/OS UNIX) processors.

shared file system
An operating system extension that allows multiple users or computers to use the same set of files at the same time, across a network. To each user, the shared file system appears to be an extension of the local file system.

shared library
On Linux and UNIX operating systems, a library that contains at least one subroutine that can be used by multiple processes.

shared library program
A program that, when loaded, is put in the shared library region for system-wide sharing.

shared library region
The area of storage in the system in which shared library objects are loaded.

shared object library
A collection of subroutines that can be shared by multiple processes.

shell
A software interface between users and an operating system. Shells generally fall into one of two categories: a command line shell, which provides a command line interface to the operating system; and a graphical shell, which provides a graphical user interface (GUI).

shell program
See shell.

shell prompt
On operating systems such as AIX or UNIX, the character string indicating that the system can accept a command. The shell prompt is typically the dollar sign ($).

shell script
A program, or script, that is interpreted by the shell of an operating system.

signal
A mechanism by which a process can be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term signal is also used to refer to the event itself.

signal handler
A subroutine or function that is called when a signal occurs.

signal mask
A collection of signals that are currently blocked from delivery to a process.

single precision
The use of one computer word to represent a number, in accordance with the required precision.

single-byte character set (SBCS)
A coded character set in which each character is represented by a 1-byte code. A 1-byte code point allows representation of up to 256 characters.

slash
The character /, also known as forward slash. This character is named <slash> in the portable character set.

SLIP
See Serial Line Internet Protocol.

SMF
See System Management Facilities.

SMF record
A collection of information about capacity and system management that is written to a Systems Management Facility (SMF) data set. Each SMF record includes information about the system's configuration, paging activity, and workload.

SMIT
See System Management Interface Tool.

socket
An identifier that an application uses to uniquely identify an end point of communication. The user associates a protocol address with the socket by associating a socket address with the socket.

software
The programs, procedures, rules, and associated documentation pertaining to the operation of a system.

sort
To rearrange some or all of a group of items, based upon the contents or characteristics of those items.

source
A system, a program within a system, or a device that makes a request to a target.

source code
A computer program in a format that is readable by people. Source code is converted into binary code that can be used by a computer.

source file
A file of programming code that is not compiled into machine language.

source language
A programming language acceptable as input to a translator.

source module
See source program.

source program
A set of instructions that are written in a programming language and must be translated into machine language before the program can be run.

source statement
A statement written in the symbols of a programming language. For example, COBOL, RPG, and DDS statements are source statements.

SP2
See Scalable Parallel 2.

space
A site intended for storage of data, such as a location in a storage medium.

spawn
A function in which a calling process (the parent process) creates a new process called a child process. The child process inherits attributes from the parent process. A new program is specified and starts running in the child process.

special character
A character other than a digit, a letter, or one of these characters: $, #, @, ., or _. For example, the following characters are special characters: *, +, and %.

special file
A file that provides an interface to input or output devices. There is at least one special file for each device attached to the computer.

square bracket
See bracket.

SRB
See service request block.

stack
An area in memory that typically stores information such as temporary register information, values of parameters, and return addresses of subroutines and is based on the principle of last in, first out (LIFO).

standard error (STDERR)
The output stream to which error messages or diagnostic messages are sent.

standard input (STDIN)
An input stream from which data is retrieved. Standard input is normally associated with the keyboard, but if redirection or piping is used, the standard input can be a file or the output from a command.

standard output (STDOUT)
The output stream to which data is directed. Standard output is normally associated with the console, but if redirection or piping is used, the standard output can be a file or the input to a command.

stanza
A group of lines in a file that together have a common function or define a part of the system. Stanzas are usually separated by blank lines or colons, and each stanza has a name.

started procedures table
A function that provides a method for assigning RACF identities to started procedures

started task
In MVS, a process that begins at system start and runs unattended. Started tasks are generally used for critical applications. The UNIX equivalent of a started task is a daemon.

statement
In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.

static storage
An area that is allocated by the system when a program is activated. Static storage exists as long as the program activation exists. If the program has not been deactivated, the values in the storage persist from one call to another.

station
A computer or device that can send or receive data.

status
The current condition or state of a program or device, for example, the status of a printer.

STDERR
See standard error

STDIN
See standard input.

STDOUT
See standard output.

sticky bit
A type of access permission bit that causes an executable program to remain on the swap area of the disk. Only someone with root authority can set the sticky bit. This bit is also used on directories to indicate that only file owners can link or unlink files in that directory.

stop
See cancel.

stopped state
A state that allows a device to be made unavailable although it is still known by the device driver, which remains loaded and bound in the kernel.

storage
The location of saved information.

storage administrator
A person in the data processing center who is responsible for defining, implementing, and maintaining storage management policies.

storage device
A physical unit that provides a mechanism to store data on a given medium so that it can be subsequently retrieved.

stream
A continuous sequence of data elements being transmitted one character at a time, or intended for transmission, using a defined format.

stream editor
A type of editor that is used to perform basic transformations on text read from a file or a pipe. The results are sent to a standard output.

structure
A class data type that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types.

stub
A protocol extension procedure that connects with the library but remains outside the library.

subcommand
A request for an operation that is within the scope of work requested by a previously issued command.

subdirectory
A directory contained within another directory in a file system hierarchy.

subprogram
A program that is called by another program, such as a subshell.

subroutine
A sequence of instructions within a larger program that performs a particular task. A subroutine can be accessed repeatedly, can be used in more than one program, and can be called at more than one point in a program.

subscript
An integer or variable whose value selects a particular element in a table or array.

subshell
An instance of the shell program started from an existing shell program.

substring
A part of a character string.

suffix
A character string attached to the end of a file name that helps identify its file type.

superuser
See root user.

superuser authority
The unrestricted ability to access and modify any part of the operating system, usually associated with the user who manages the system.

supervisor
The part of a control program that coordinates the use of resources and maintains the flow of processor operations.

supplementary group ID
A process attribute that is used when file access permissions are determined.

symbol table
A list of symbol names and their associated values, usually in an object or executable file, which gives the names of external symbols and their addresses.

symbolic link
A type of file that contains a pointer to another file or directory.

synchronous
Occurring with a regular or predictable time relationship.

synchronous transmission
A method of transmission in which the sending and receiving of data is controlled by timing signals.

syntax
The rules for the construction of a command or statement.

sysplex
A set of z/OS systems that communicate with each other through certain multisystem hardware components and software services.

sysplex-aware
In zFS, pertaining to a physical file system that handles file requests for mounted file systems locally instead of shipping function requests through z/OS UNIX.

sysplex CDS
See sysplex couple data set.

sysplex couple data set (sysplex CDS)
A couple data set (CDS) that contains sysplex-wide data about systems, groups, and members that use cross-system coupling facility (XCF) services. All systems in a sysplex must be connected to the sysplex CDS.

Sysplex Timer
An IBM unit that synchronizes the time-of-day (TOD) clocks in processors.

system
A computer and its associated devices and programs.

system administrator (SA)
The person who controls and manages a computer system.

System Authorization Facility (SAF)
An MVS interface with which programs can communicate with an external security manager, such as RACF.

System Display and Search Facility (SDSF)
An IBM-licensed program that provides a menu-driven full-screen interface that is used to obtained detailed information about jobs and resources in a system.

system dump (SDUMP)
A dump of all the storage in the system that can be used for problem determination.

System Management Facilities (SMF)
A component of z/OS that collects and records a variety of system and job-related information.

System Management Interface Tool (SMIT)
An interface tool of the AIX operating system for installing, maintaining, configuring, and diagnosing tasks.

system program
A program providing services in general support of the running of a system.

system programmer
A programmer who plans, maintains, and controls the use of an operating system with the aim of improving overall productivity of an installation.

tag
A mechanism used to identify certain attributes having some bearing on handling of character data. Some examples are character set identifier, code page identifier, language identifier, country identifier, and encoding scheme identifier.

target
The program or system to which a request for files or processing is sent.

task
A unit of work to be accomplished by a device or process.

task control block (TCB)
A z/OS control block that is used to communicate information about tasks within an address space that is connected to a subsystem.

TCB
See task control block.

TCP
See Transmission Control Protocol.

TCP/IP
See Transmission Control Protocol/Internet Protocol.

tcsh
See Tenex C shell.

temporary file system (TFS)
A temporary, in-memory physical file system that supports in-storage mountable file systems. Normally, a TFS runs in the kernel address space, but it can be run in a logical file system (LFS) colony address space.

temporary storage
The section of computer storage in which data is stored temporarily while a program is running.

Tenex C shell (tcsh)
An enhancement of the UNIX C shell (csh) that is compatible with csh.

term
The smallest part of an expression that can be assigned a value.

terminal
In data communication, a device, usually equipped with a keyboard and display device, capable of sending and receiving information.

terminal device file
See character special file.

Terminal Monitor Program (TMP)
The program that manages a Time Sharing Option (TSO) session.

terminal type (tty)
A generic device driver for a text display. A tty typically performs input and output on a character-by-character basis.

text editor
A program used to create, modify, and print or display text files.

text file
A file that contains only printable characters.

TFS
See temporary file system.

thread
A stream of computer instructions that is in control of a process. In some operating systems, a thread is the smallest unit of operation in a process. Several threads can run concurrently, performing different jobs.

tilde
One of the accent marks in Latin script (~).

Time Sharing Option (TSO)
A base element of the z/OS operating system with which users can interactively work with the system.

Time Sharing Option Extensions (TSO/E)
A licensed program that is based on Time Sharing Option (TSO). With TSO/E, MVS users can interactively share computer time and resources.

timestamp
The value of an object that indicates the system time at some critical point in the object's history.

timeout
A time interval that is allotted for an event to occur or complete before operation is interrupted.

TMP
See Terminal Monitor Program.

token
The basic syntactic unit of a computing language. A token consists of one or more characters, excluding the blank character and excluding characters within a string constant or delimited identifier.

token number
A nonnegative integer that represents the name of a token.

touch
To set a flag in a window that indicates that the information in the window could differ from the that displayed on the terminal device.

trace
To record data that provides a history of events occurring in the system.

track
A circular path on the surface of a disk or diskette on which information is magnetically recorded and from which recorded information is read.

transactional
Pertaining to an application program that is divided into segments, where each segment typically requests an I/O operation with a terminal user, giving up control to other application program segments for the duration of the I/O operation.

transient
Pertaining to a program or subroutine that does not reside in main storage.

transmission
The sending of data from one place for reception elsewhere.

Transmission Control Protocol (TCP)
A communication protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in interconnected systems of such networks.

Transmission Control Protocol/Internet Protocol (TCP/IP)
An industry-standard, nonproprietary set of communication protocols that provides reliable end-to-end connections between applications over interconnected networks of different types.

trap
A special statement used to catch signals within the z/OS shell.[OSF]

truncate
To shorten a field, value, statement, or string.

TSO
See Time Sharing Option.

TSO/E
See Time Sharing Option Extensions.

tty
See terminal type.

tuning
The process of adjusting an application, a system, or system control variables to operate in a more efficient manner.

UI
See user interface.

underscore character
A character used in each position of an entry field to indicate its length. This indicator of entry field length is used on display devices that do not have the underscore attribute.

undub
To make an address space unknown to MVS.

unformatted file
A file that is arranged without such characteristics as a certain number of characters and lines per page, line spacing, and headings.

union
A variable that can hold any one of several data types, one data type at a time.

union tag
An identifier that names a union data type.

UNIX
A highly portable operating system that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers, but was adapted for mainframes and microcomputers. The AIX operating system is IBM's implementation of the UNIX operating system.

UNIX file
An object that exists in a hierarchical file system. Examples of UNIX files are zFS, ZFS, NFS, and TFS.

UNIX System Services
An element of z/OS that creates a UNIX environment that conforms to XPG4 UNIX 1995 specifications and that provides two open-system interfaces on the z/OS operating system: an application programming interface (API) and an interactive shell interface.

unmount
To logically disassociate a mountable file system from another file system.

user
Any individual, organization, process, device, program, protocol, or system that uses the services of a computing system.

user address space
An address space that has at least one MVS task known to the kernel address space. This address space can contain a shell or an application program that uses UNIX System Services.

user area
The parts of main storage and disk available to the user.

user ID
See user identification.

user identification (user ID)
The name used to associate the user profile with a user when a user signs on to a system.

user interface (UI)
The hardware, or software, or both that enables a user to interact with a system, program, or device.

user name
A string of characters that uniquely identifies a user to a system.

user profile
In computer security, a description of a user that includes such information as user ID, user name, password, access authority, and other attributes that are obtained when the user logs on.

UTC
See Coordinated Universal Time.

UTF-8
Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208.

valid
Pertaining to that which is allowed, is true, or conforms to some standard.

value
In programming, the alphabetic or numeric contents of a variable or a storage location.

variable
A representation of a changeable value.

variable-length record
A record having a length independent of the length of other records with which it is logically or physically associated.

vector
An array of one dimension.

version file system
See root file system.

VFS
See virtual file system.

virtual file system (VFS)
A remote file system that has been mounted so that it is accessible to the local user.

virtual lookaside facility (VLF)
A z/OS facility that enables named data to be kept in virtual storage instead of DASD.

virtual storage (VS)
The storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped to real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations.

Virtual Storage Access Method (VSAM)
An access method for direct or sequential processing of fixed-length and variable-length records on disk devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative-record number.

Virtual Telecommunications Access Method (VTAM)
An IBM licensed program that controls communication and the flow of data in an SNA network.

VLF
See virtual lookaside facility.

volume
A discrete unit of storage on disk, tape or other data recording medium that supports some form of identifier and parameter list, such as a volume label or input/output control.

VS
See virtual storage.

VSAM
See Virtual Storage Access Method.

VTAM
See Virtual Telecommunications Access Method.

wait
A state allowing a parent process to synchronize with the execution of an exit issued by a child process.

white space
A sequence of one or more characters, such as the blank character, the newline character, or the tab character, that belong to the space character class.

wide character
A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales.

wildcard character
A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace the wildcard character.

window
An area of the screen with visible boundaries in which an application program or information is displayed or in which a dialog is presented.

word
A fundamental unit of storage that refers to the amount of data that can be processed at a time. Word size is a characteristic of the computer architecture. See also halfword.

work area
That portion of central storage that is used by a computer program to hold data temporarily.

working directory
The active directory. When a file name is specified without a directory, the current directory is searched.

working storage
See temporary storage.

workstation
A terminal or microcomputer at which a user can run applications and that is usually connected to a mainframe or a network.

write
To output characters to a file, such as standard output or standard error. Unless otherwise stated, standard output is the default output destination for all uses of the term write. [POSIX.2]

write access
In computer security, permission to write to an object.

write lock
A lock that prevents any other process from setting a read lock or a write lock on any part of the protected area.

write to log (WTL)
A system service used to send messages to the system log or hardcopy log.

write to operator with reply (WTOR)
A system service used to send messages to an operator console informing the operator of errors and system conditions that might need correcting. A response is required.

WTL
See write to log.

WTOR
See write to operator with reply.

X Window System
A software system, developed by the Massachusetts Institute of Technology, that enables the user of a display to concurrently use multiple application programs through different windows of the display. The application programs can execute on different computers.

XCF
See cross-system coupling facility.

XCF couple data set
A data set that is created through the cross-system coupling facility (XCF) couple data set (CDS) format utility and, depending on its designated type, is shared by some or all of the systems in a sysplex.

XPLINK
See Extra Performance Linkage.

z/OS
An IBM mainframe operating system that uses 64-bit real storage.

z/VM®
An IBM mainframe operating system that acts as a hypervisor. z/VM can virtualize all system resources, including processors, memory, storage devices, communication devices, and networking, and can dynamically add or increase system resources. z/VM supports the concurrent operation of hundreds of operating system instances.

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

Product Number: 5655-ZOS

GA32-0884-60