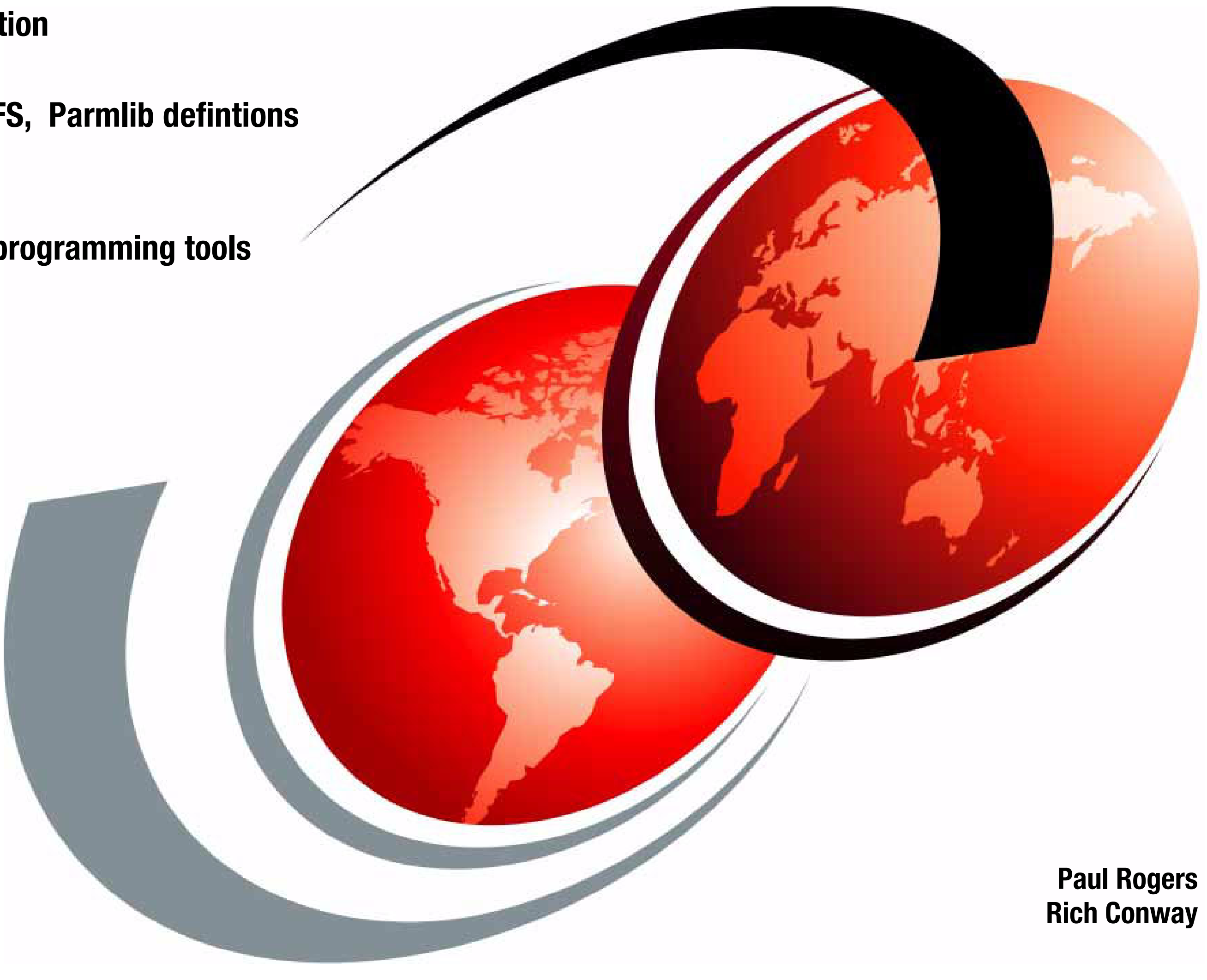
IBM

ABCs of z/OS System Programming Volume 9

z/OS UNIX, TCP/IP installation, customization

HFS and zFS, Parmlib defintions

Shell and programming tools



Paul Rogers
Rich Conway

Redbooks

ibm.com/redbooks

International Technical Support Organization

ABCs of z/OS System Programming Volume 9

December 2003

SG24-6989-00

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

First Edition (December 2003)

This edition applies to Version 1 Release 4 of z/OS (5694-A01), to Version 1 Release 4 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Notices

Trademarks

Preface

The ABCs of z/OS® System Programming is a ten volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

This redbook describes UNIX® System Services (z/OS UNIX). It will help you install, tailor, configure, and use the z/OS Version 1 Release 4 version of z/OS UNIX. Topics covered in this volume are the following:

- The products and components used with z/OS UNIX
- An overview of z/OS UNIX
- z/OS UNIX pre-installation requirements
- Step-by-step installation of UNIX System Services using the ServerPac
- The z/OS UNIX shell and utilities
- z/OS UNIX security customization using RACF®
- The Hierarchical File System (HFS) and zFS
- TCP/IP overview and customization; TCP/IP applications
- z/OS UNIX parmlib Member definitions
- How to install maintenance
- Using z/OS UNIX operator commands
- The z/OS UNIX shell and programming tools
- Using the workload manager; performance and tuning considerations
- z/OS UNIX printing options

The contents of the other volumes are as follows:

- Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, MVS™ delivery and installation
- Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, catalogs
- Volume 3: Introduction to DFSMS, storage management
- Volume 4: Communication Server, TCP/IP, and VTAM®
- Volume 5: Base and Parallel Sysplex®, system logger, global resource serialization, z/OS system operations, automatic restart management, hardware management console, performance management
- Volume 6: RACF, PKI, LDAP, cryptography, Kerberos, and firewall technologies
- Volume 7: Infoprint® Server, Language Environment®, and SMP/E
- Volume 8: z/OS problem diagnosis
- Volume 10: Introduction to z/Architecture™, zSeries® processor design, zSeries connectivity, LPAR concepts, and HCD

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Paul Rogers is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM® classes worldwide on various aspects of z/OS and OS/390®. Before joining the ITSO 15 years ago, Paul worked in the IBM Installation Support Center (ISC) in Greenford, England providing OS/390 and JES support for IBM EMEA and the Washington Systems Center. He has worked for IBM for 36 years.

Rich Conway is a systems programmer at the International Technical Support Organization, Poughkeepsie Center. He has extensive knowledge of UNIX System Services and all of the products installed at the ITSO. He has worked in the ITSO for the last 11 years.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Products and components

This chapter provides a brief overview of UNIX System Services (z/OS UNIX) and related components. z/OS UNIX interacts with the following elements and features of z/OS:

- BCP (WLM and SMF components).
- C/C++ Compiler, to compile programs.
- Language Environment, to execute the shell and utilities or any other.
- XPG4-compliant shell application.
- Data Facility Storage Management Subsystem/MVS (DFSMS/MVS®). HFS is a component of DFSMS/MVS.
- Security Server for z/OS. (RACF is a component of the Security Server.)
- Resource Measurement Facility (RMF™).
- System Display and Search Facility (SDSF).
- Time Sharing Option Extensions (TSO/E).
- z/OS Communications Server (TCP/IP).
- ISPF, to use the dialogs for OEDIT, or ISPF/PDF for the ISPF shell.
- BookManager® READ/MVS, to use the OHELP online help facility.
- Network File System (NFS).
- z/OS Distributed File Service zSeries File System (zFS).

1.1 UNIX System Services

- A new name for OpenEdition
  - Abbreviation - OS/390 UNIX
    - Beginning with OS/390 Release 5
  - New abbreviation - z/OS UNIX
    - Beginning with z/OS Version 1 Release 1

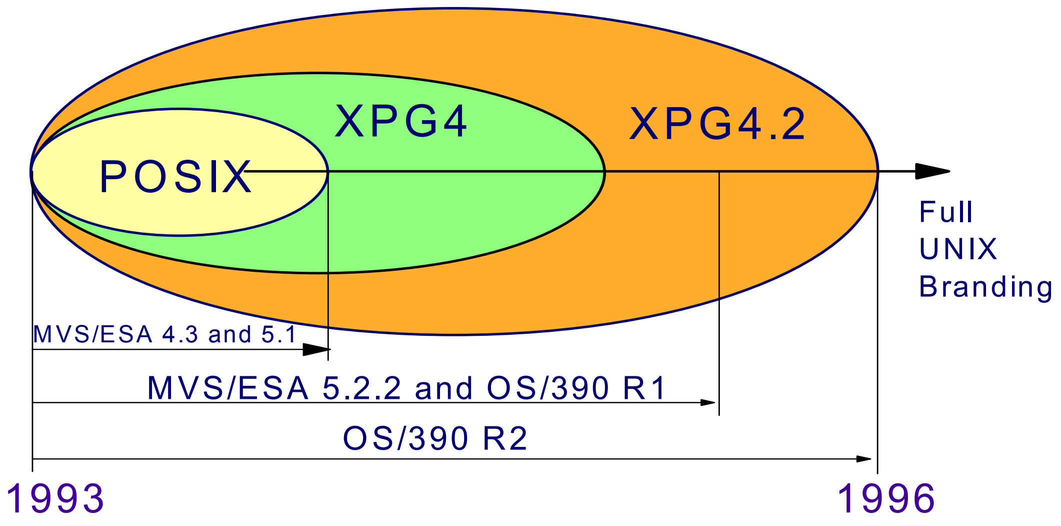


Figure 1-1 Implementation of UNIX System Services

The name OpenEdition® was changed to OS/390 UNIX System Services beginning with OS/390 Release 5. UNIX Services can then be abbreviated OS/390 UNIX.

When OS/390 was renamed to z/OS, the new abbreviation for UNIX System Services became z/OS UNIX.

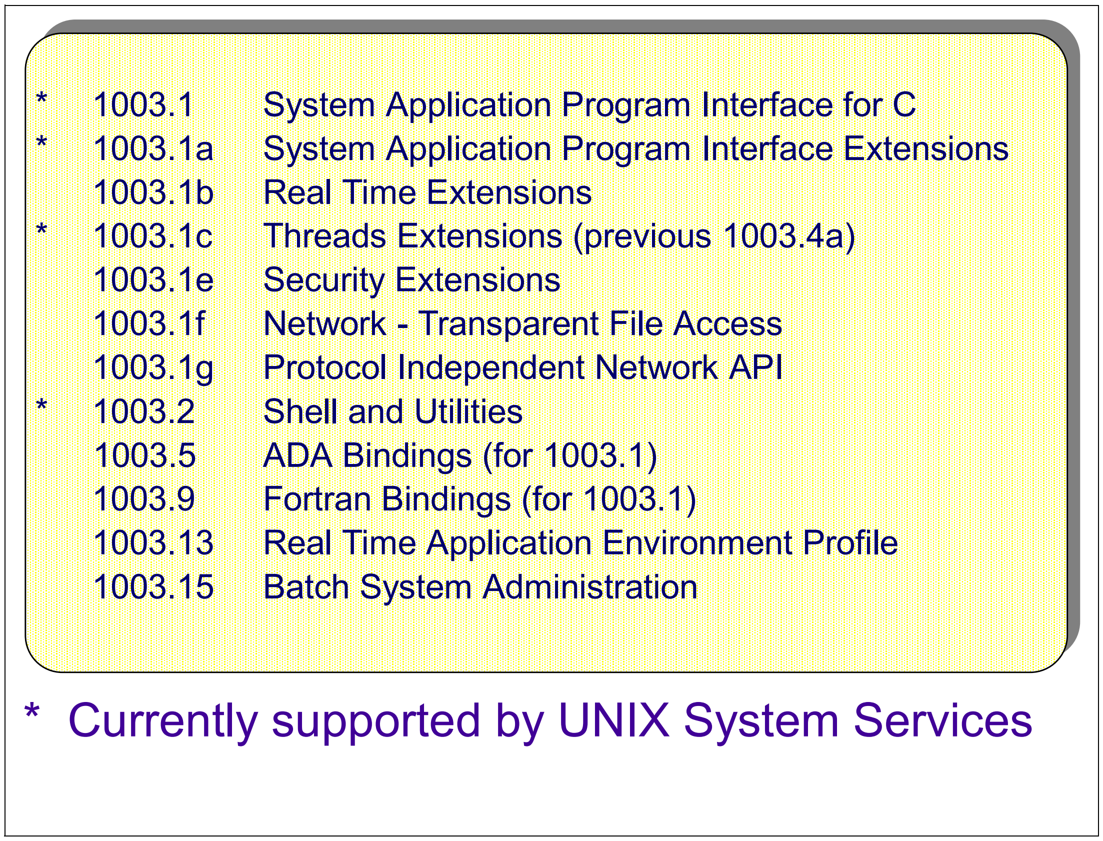
z/OS UNIX was originally implemented in MVS/ESA™ 4.3 as OpenEdition and supported the POSIX standards (1003.1, 1003.1a, 1003.1c, and 1003.2) with approximately 300 functions. In MVS/ESA 5.2.2 many additional functions were added to meet the XPG4 requirements.

In MVS/ESA 5.2.2 more than 1100 functions were included in the OpenEdition implementation. This incorporated the full X/Open Portability Guide issue 4 (XPG4) and over 90% of the single UNIX specification as defined in XPG4.2. The remaining functions were added afterwards, and OpenEdition became branded as a UNIX system.

The XPG4.2 support includes all commands and utilities, most of the additional C services defined in the standard and curses, which was included in specification 1170 but not in the XPG4.2 itself. Curses is the UNIX multicolor, multilanguage screen control package which comes from the Novell SVID Edition 3 package.

Since OS/390 V2R2 the following items were added: STREAMS, X/Open Transport Interface (XTI), XPG4.2 regular expressions, XPG4.2 context switching, and XPG4.2 behavior specific to sockets.

1.2 POSIX standards overview



*	1003.1	System Application Program Interface for C
*	1003.1a	System Application Program Interface Extensions
	1003.1b	Real Time Extensions
*	1003.1c	Threads Extensions (previous 1003.4a)
	1003.1e	Security Extensions
	1003.1f	Network - Transparent File Access
	1003.1g	Protocol Independent Network API
*	1003.2	Shell and Utilities
	1003.5	ADA Bindings (for 1003.1)
	1003.9	Fortran Bindings (for 1003.1)
	1003.13	Real Time Application Environment Profile
	1003.15	Batch System Administration

* Currently supported by UNIX System Services

Figure 1-2 Overview of POSIX standards

The work on Portability Operating Systems Interface (POSIX) started as an effort to standardize UNIX and was performed by a workgroup under the Institute of Electrical and Electronics Engineers (IEEE). What they defined was an application programming interface which could be applied not only to UNIX systems but to other operating systems like MVS.

POSIX is not a product. It is an evolving family of standards describing a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.

The POSIX standard is sponsored by the International Organization for Standardization (ISO) and is incorporated into X/Open Portability Guides (XPG). Each element of the standard is defined by a 1003.* number.

POSIX defines the *interfaces* and not the solution or implementation. In this way POSIX can be supported by any operating system. Implementation of POSIX can be different in areas such as performance, availability, and recoverability. All POSIX-compliant systems aren't the same, although they all support basically the same interface.

The support for open systems in z/OS is based on the POSIX standard.

1.3 z/OS Operating System and z/OS UNIX

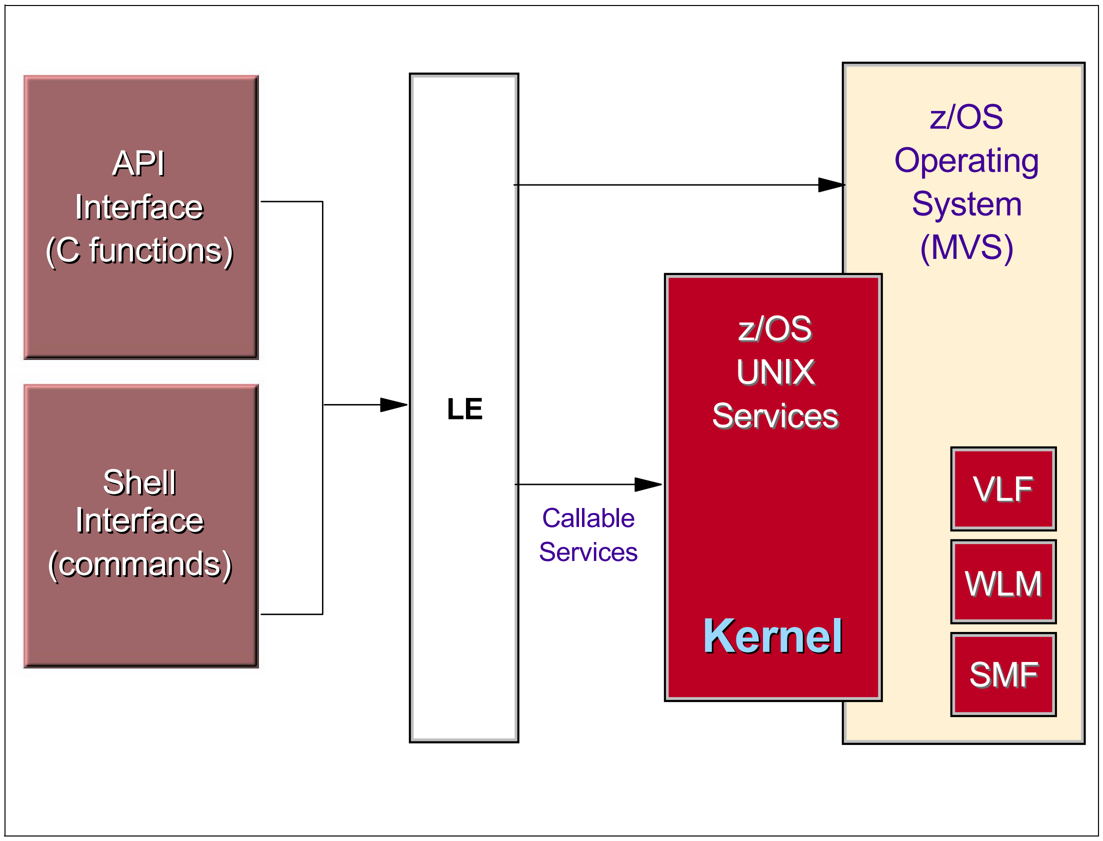

API Interface (C functions)
Shell Interface (commands)
LE
Callable Services
z/OS UNIX Services
Kernel
z/OS Operating System (MVS)
VLF
WLM
SMF

Figure 1-3 z/OS UNIX and the z/OS operating system

The z/OS support for UNIX System Services (z/OS UNIX) enables two open systems interfaces on the z/OS operating system:

- An application program interface (API)
- An interactive z/OS shell interface

With the APIs, programs can run in any environment—including in batch jobs, in jobs submitted by TSO/E users, and in most other started tasks—or in any other MVS application task environment. The programs can request:

- Only MVS services
- Only z/OS UNIX
- Both MVS and z/OS UNIX

The 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

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 the Language Environment (LE).

With the z/OS UNIX System Services Application Services, users can:

- Request services from the system through shell commands. Shell commands are like TSO/E commands.
- Write shell scripts to run tasks. Shell scripts are analogous to REXX EXECs.
- Run programs interactively (in the foreground) or in the background.

Many users use similar interfaces on other systems, such as AIX® for the RISC System/6000® or UNIX, 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 a z/OS system.

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

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

A z/OS UNIX program can be run interactively from a shell in the foreground or background, run as a z/OS batch job, or called from another program. The following types of applications exist in a z/OS system with z/OS UNIX:

- Strictly conforming XPG4 UNIX-conforming applications
- Applications using only kernel services
- Applications using both kernel and z/OS services
- Applications using only z/OS 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, the system dubs the program as a z/OS UNIX process, unless it is a POSIX(OFF) program (which is not dubbed).

1.4 Product and component support for z/OS UNIX

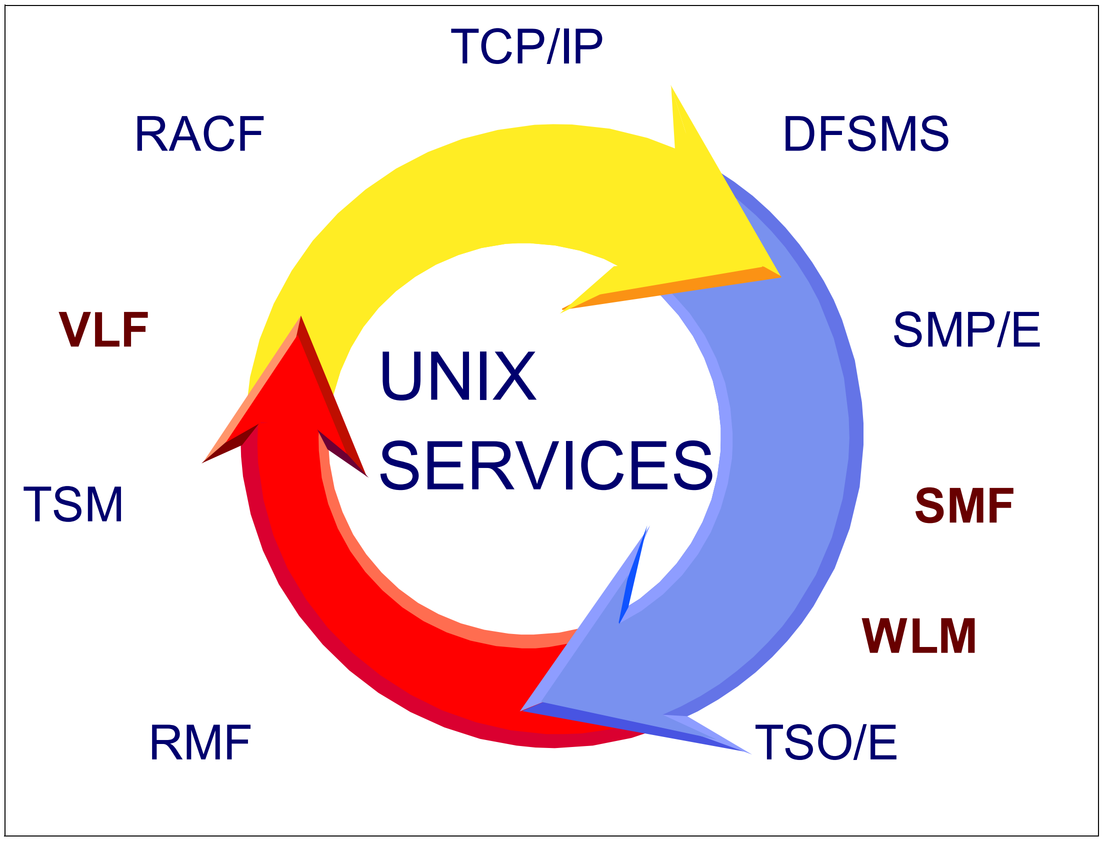


Figure 1-4 Products and components used by z/OS UNIX

The rest of this chapter provides an introduction to the products (RACF, TCP/IP, DFSMS, TSM, RMF, TSO/E, and SMP/E) and components (VLF, SMF, and WLM) used with UNIX System Services.

z/OS components:

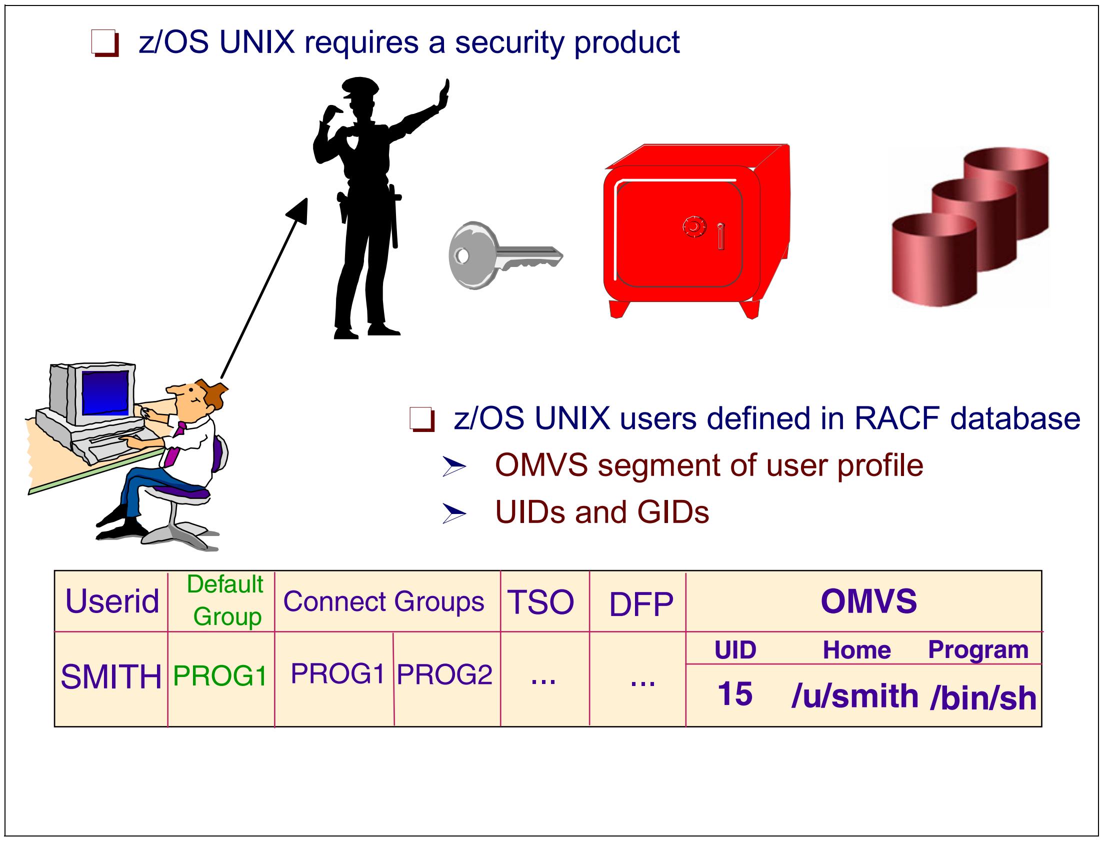
WLM The workload manager (WLM) transaction initiators provide address spaces when programs issue the fork(), spawn(), C function, or z/OS callable services.

VLF RACF allows caching of UID and GID information in the Virtual Lookaside Facility (VLF). Add the following VLF options to the COFVLFxx member of SYS1.PARMLIB to enable caching since it will improve the performance of z/OS UNIX. VLF is still valid if you have not converted to AIM stage 3.

```
 CLASS NAME(IRRUMAP)
     EMAJ(UMAP)
CLASS NAME(IRRGMAP)
     EMAJ(GMAP)
```

SMF 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 on resources used by the process. SMF File System records describe file system events such as file open, file close, file system mount, unmount, quiesce, and unquiesce.

1.5 Security Server RACF



- z/OS UNIX requires a security product
- z/OS UNIX users defined in RACF database
  - OMVS segment of user profile
  - UIDs and GIDs

Userid	Default Group	Connect Groups		TSO	DFP	OMVS		
						UID	Home	Program
SMITH	PROG1	PROG1	PROG2	...	...	15	/u/smith	/bin/sh

Figure 1-5 Security Server RACF

RACF or an equivalent security product like ACF2 can manage system and data set security by verifying a user and checking that the user can access a resource.

The security products also take care of who is allowed to issue special commands, define new users, and change access permissions.

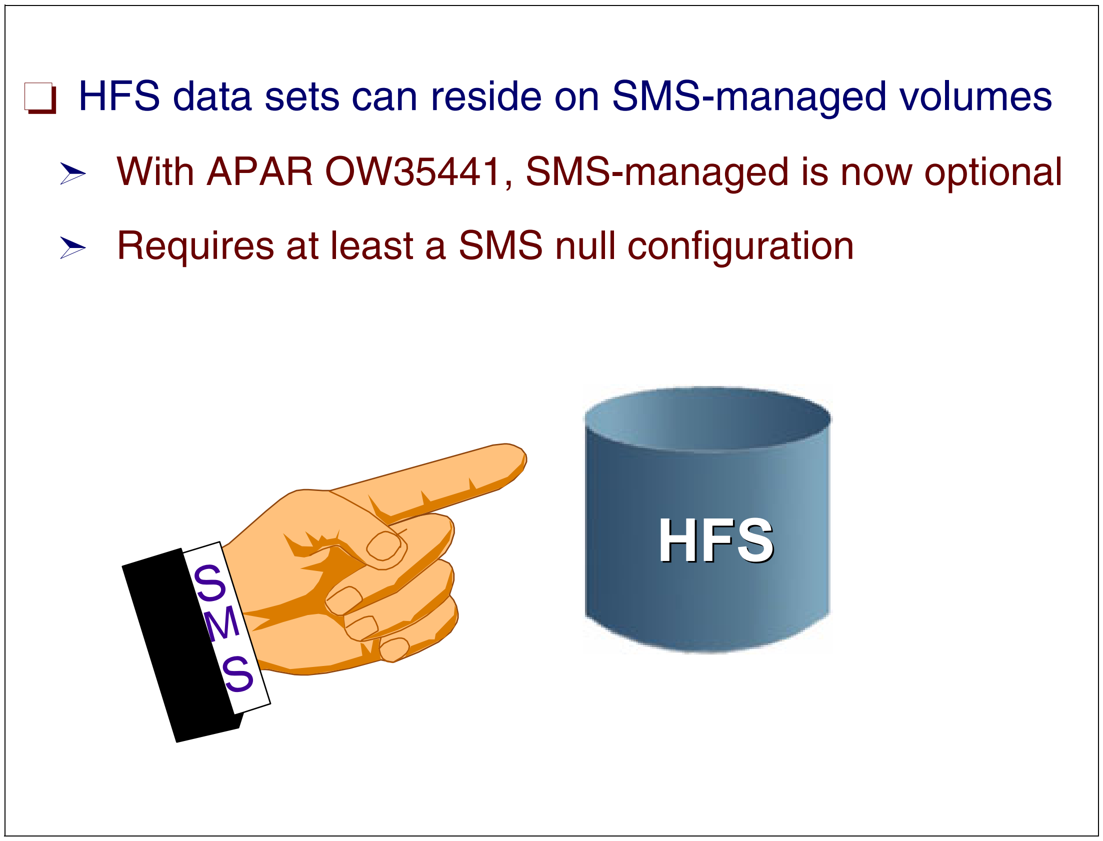
The permission to access directories and files is only UNIX related. RACF is responsible for the OMVS segment which allows the user to access UNIX Services.

In other words, if someone wants to access UNIX file systems, whether a directory or a file, RACF checks whether this user is allowed to have access in the z/OS UNIX environment. The security checking to access specific directories or files is then handled by UNIX System Services itself.

RACF is not able to change permission bits or owner statements in the UNIX environment. RACF only checks whether a user or program is allowed to access UNIX services.

The RACF user profile definition was expanded with a segment called OMVS for z/OS UNIX support. All users and programs that need access to z/OS UNIX must have a RACF user profile defined with an OMVS segment which has, as a minimum, a UID specified. A user without a UID cannot access z/OS UNIX.

1.6 Data Facility System-Managed Storage (DFSMS)


❑ HFS data sets can reside on SMS-managed volumes

➢ With APAR OW35441, SMS-managed is now optional

➢ Requires at least a SMS null configuration

Figure 1-6 DFSMS and z/OS UNIX

Data Facility System-Managed Storage (DFSMS) manages the z/OS UNIX data sets used for processing. These hierarchical file system (HFS) data sets make up a file hierarchy.

DFSMS manages the hierarchical file system (HFS) data sets that contain the file systems. To use kernel services in full-function mode, SMS must be active.

A hierarchical file system consists of:

- Files, which 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, which contain files, other directories, or both.
- Additional local or remote file systems, which are mounted within the file hierarchy.

1.7 Transmission Control Protocol/Internet Protocol (TCP/IP)

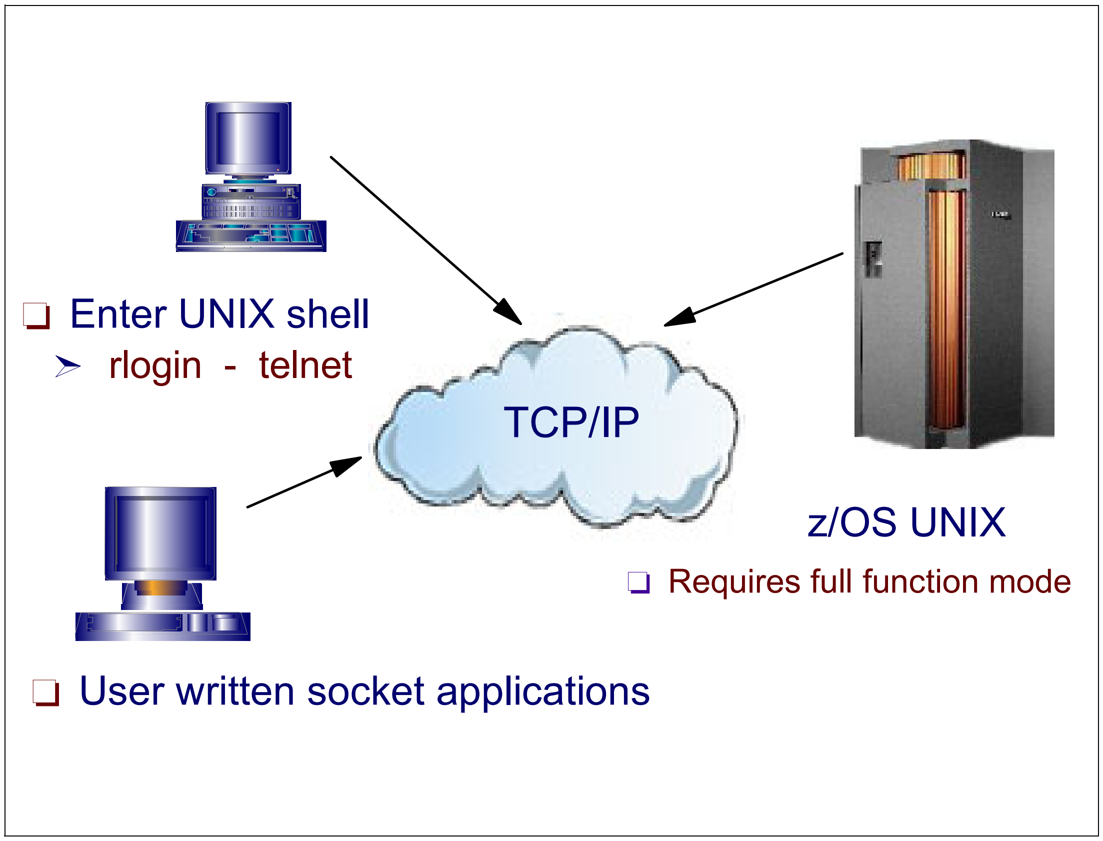

❑ Enter UNIX shell
➢ rlogin - telnet
TCP/IP
z/OS UNIX
❑ Requires full function mode
❑ User written socket applications

Figure 1-7 TCP/IP and z/OS UNIX

Transmission Control Protocol/Internet Protocol is a peer-to-peer network protocol. It is officially named the TCP/IP Internet Protocol Suite and is referred to as TCP/IP (after the name of its two main standard protocols). TCP/IP is a set of industry standard protocols and applications.

The TCP/IP Internet Protocol Suite is a layered set of protocols which allow cooperating computers to share resources across a network or networks. The physical networks may be of different types. A way to enter the shell environment is by using rlogin or telnet from a workstation in the TCP/IP network.

User-written socket applications can use TCP/IP as a communication vehicle. Both client and server socket applications can use the socket interface to communicate over the Internet (AF_INET) 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.

The OMVS= parameter in the IEASYSxx parmlib member specifies the BPXPRMxx member or members that determine the configuration of the kernel service. Minimum mode provides the minimum requirements for kernel services; you will not be able to use the shell or TCP/IP in this mode. If you do not specify OMVS= in the IEASYSxx parmlib member, or if you specify OMVS=DEFAULT, then kernel services start up in a minimum configuration mode with all BPXPRMxx parmlib options taking their default values when the system is IPLed. This mode is for installations that do not plan to use the kernel services.

1.8 System Modification Program Extended (SMP/E)

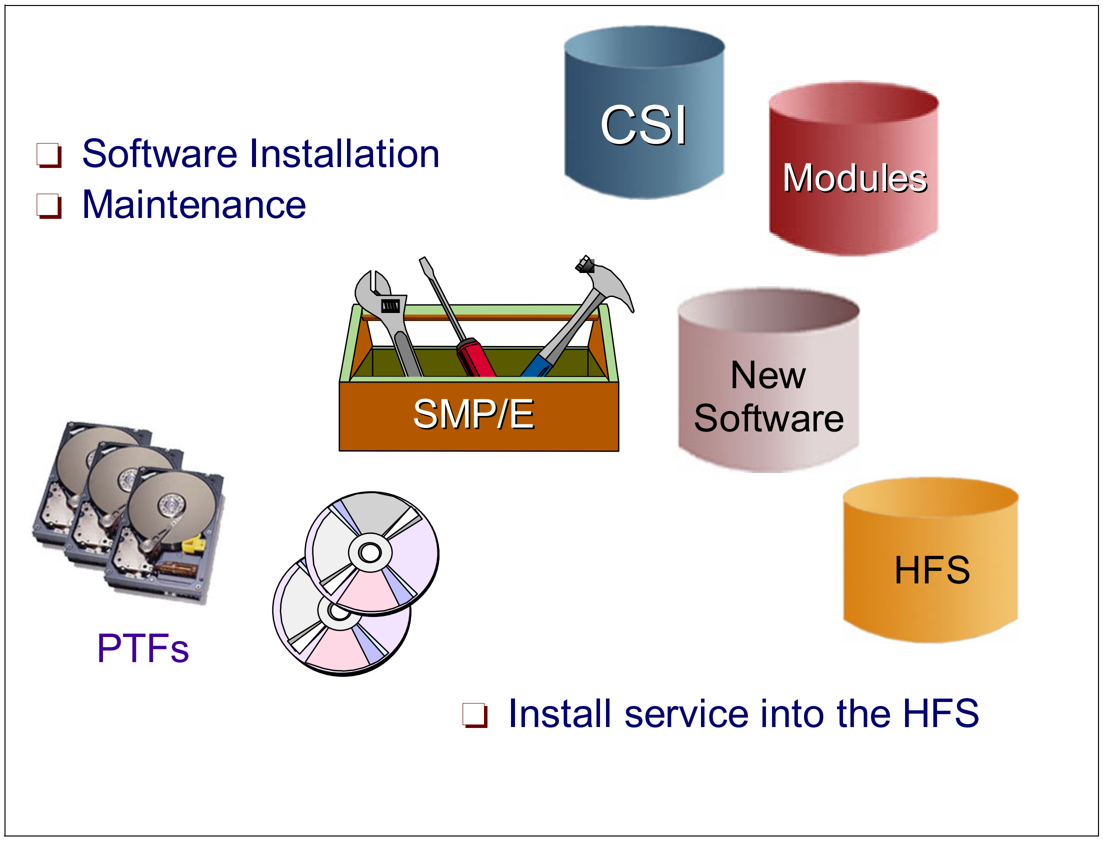


Figure 1-8 SMP/E and z/OS UNIX

SMP/E is a basic tool for installing and maintaining software in MVS systems and subsystems. It controls these changes by:

- Selecting the proper levels of elements to be installed from a large number of potential changes.
- Calling system utility programs to install the changes.
- Keeping records of the installed changes (in the CSI).

SMP/E is an integral part of the installation, service, and maintenance processes for CBIPOs, CBPDOs, ProductPacs, ServicePacs, and selective follow-on service for CustomPacs. In addition, SMP/E can be used to install and service any software that is packed in SMP/E system modification (SYSMOD) format.

SMP/E can be run either using batch jobs or using dialogs under Interactive System Productivity Facility/Program Development Facility (ISPF/PDF).

Two types of dialogs are provided by SMP/E:

- CBIPO dialogs for installing and redistributing Custom-Built Installation Process Offering (CBIPO) packages.
- SMP/E dialogs help you interactively query the SMP/E database, as well as create and submit jobs to process SMP/E commands.

1.9 System Management Facility (SMF)

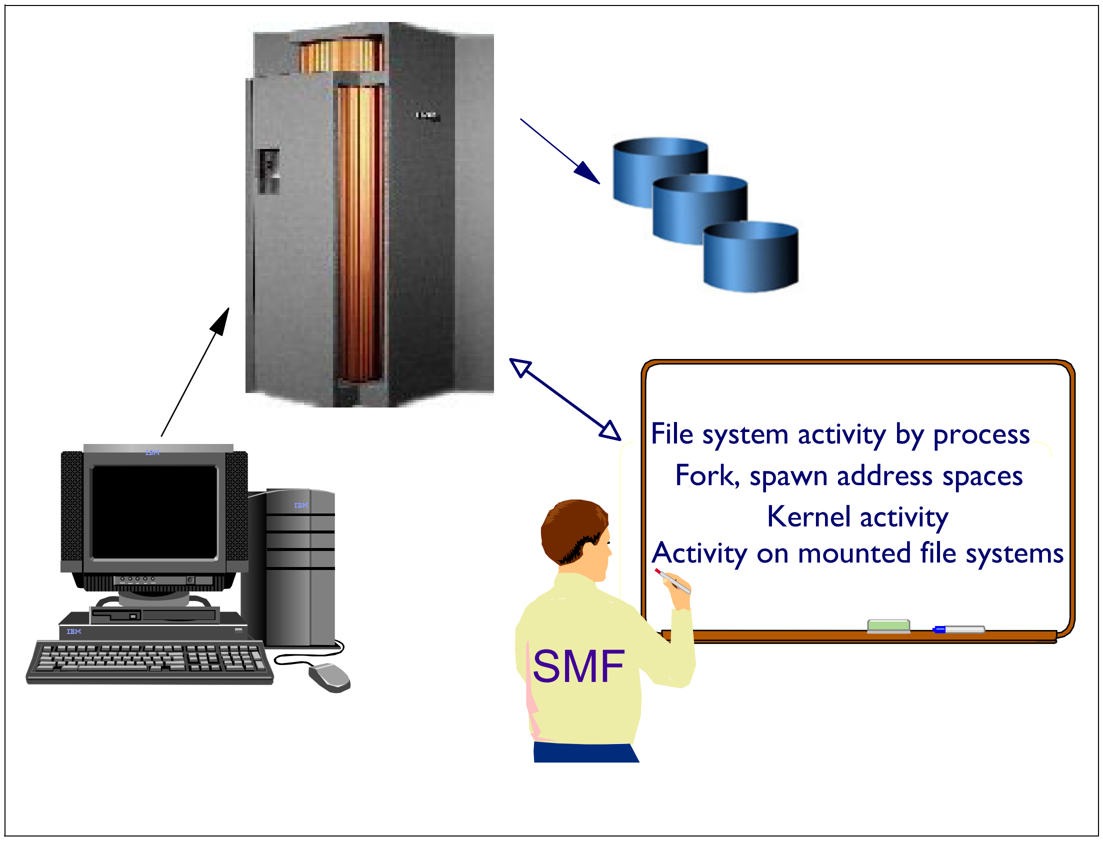

File system activity by process
Fork, spawn address spaces
Kernel activity
Activity on mounted file systems
SMF

Figure 1-9 SMF and z/OS UNIX

System Management Facility (SMF), which is a component of the BCP element, 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 on 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.

SMF job and job-step accounting records identify z/OS UNIX processes by:

- User
- Process
- Group
- Session identifiers

1.10 Resource Measurement Facility (RMF)

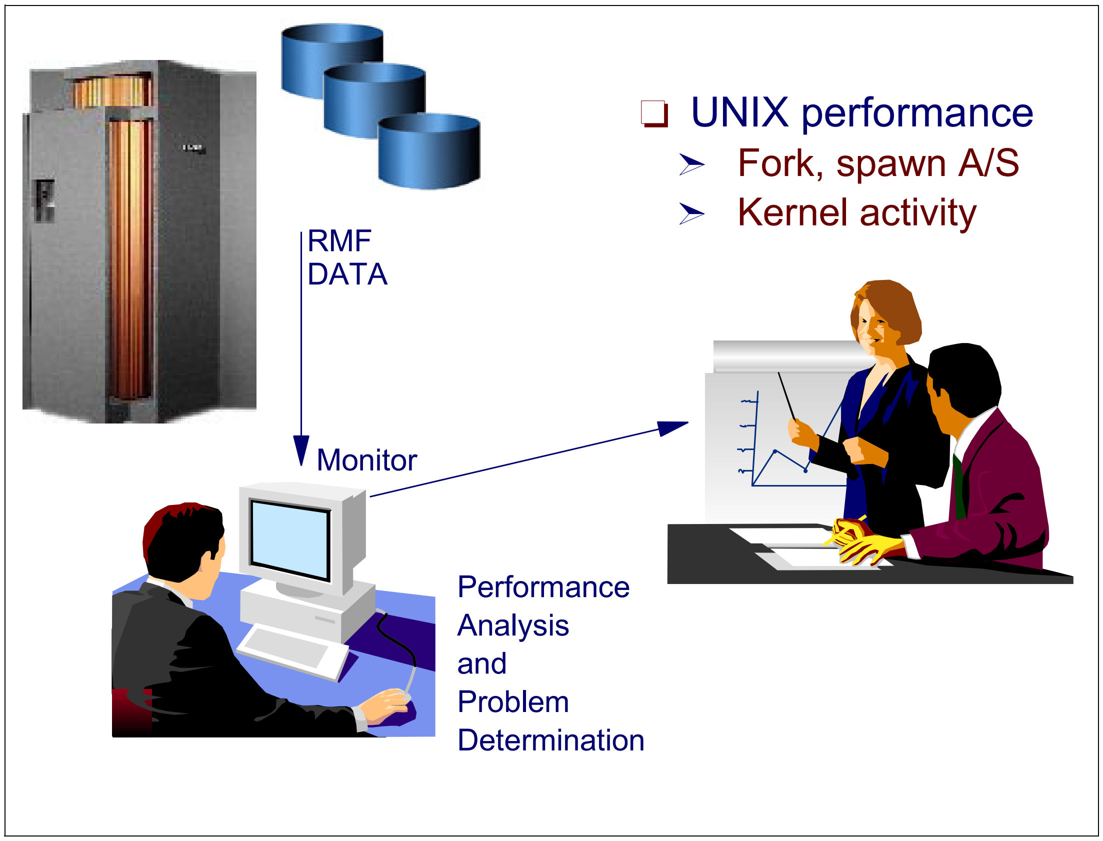


Figure 1-10 RMF and z/OS UNIX

The software products supporting system programmers and operators in managing their systems heavily influence the complexity of their jobs, and their ability to keep systems at a high level of availability.

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.

When an installation specifies an OMVS subsystem type in the workload manager service policy, RMF shows the activity of forked address spaces separately in the RMF Workload Activity report.

RMF monitors the use of resources in an OMVS Kernel Activity report.

1.11 Virtual Lookaside Facility (VLF)

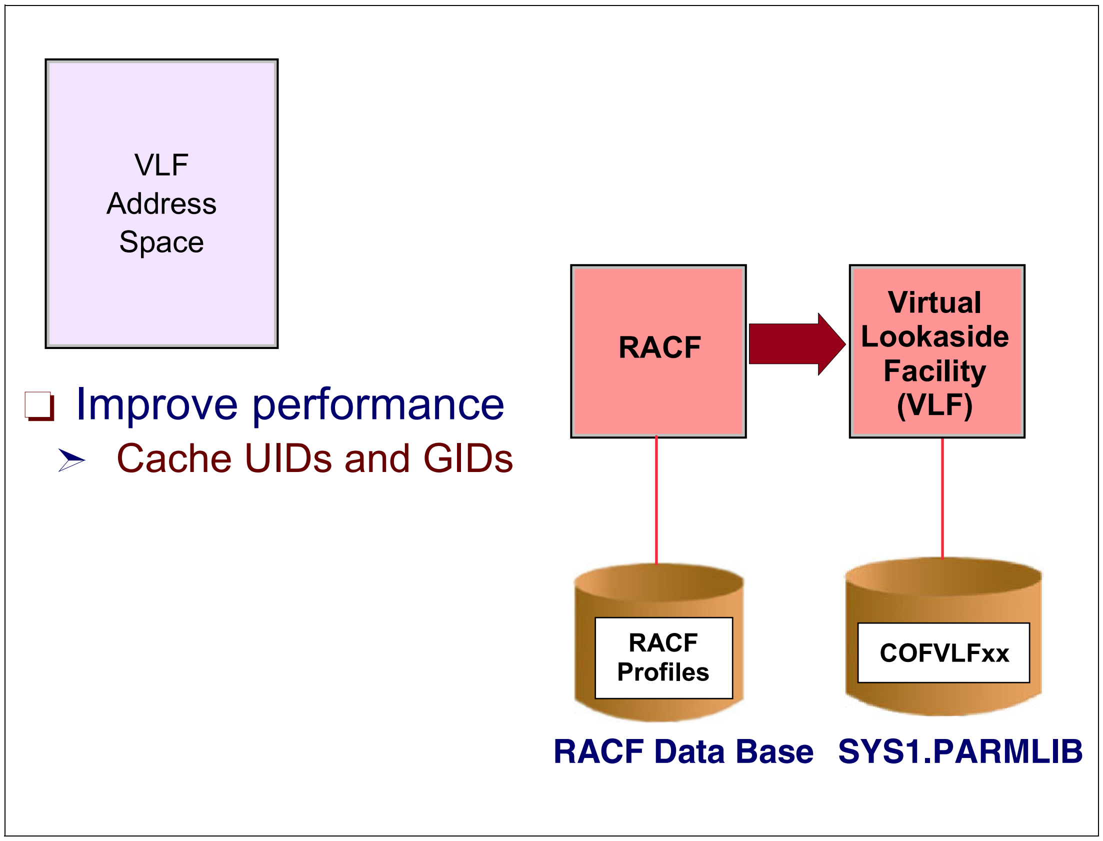


Figure 1-11 VLF and z/OS UNIX

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. 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 should 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.

The Virtual Lookaside Facility is a set of easy-to-use high-performance services that provide an alternate fast path method for making frequently used named data objects available to many users. It will reduce I/O operations for frequently accessed programs that are provided now in central or expanded storage.

Whether you are developing a new application or modifying an existing one, the kind of application that can use VLF effectively is an application that frequently retrieves a repetitive set of data from DASD on behalf of many end users.

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 VLF. Add the following VLF options to the COFVLFxx member of SYS1.PARMLIB to enable the caching:

```
CLASS NAME(IRRUMAP)
  EMAJ(UMAP)
```

```
CLASS NAME(IRRGMAP)
  EMAJ(GMAP)
CLASS NAME(IRRGTS)
  EMAJ(GTS)
CLASS NAME(IRRACEE)
  EMAJ(ACEE)
CLASS NAME(IRRSMAP)
  EMAJ(SMAP)
```

For details about the VLF classes, see *z/OS Security Server RACF System Programmer's Guide,* SA22-7681.

AIM stage 3

In stage 3, RACF locates application identities, such as UIDs and GIDs, for users and groups by using an alias index that is automatically maintained by RACF. This allows RACF to more efficiently handle authentication and authorization requests from applications such as z/OS UNIX than was possible using other methods, such as the UNIXMAP class and VLF. Once your installation reaches stage 3 of application identity mapping (AIM), you will no longer have UNIXMAP class profiles on your system, and you can deactivate the UNIXMAP class and remove VLF classes IRRUMAP and IRRGMAP.

Important: Associating RACF user IDs and groups to UIDs and GIDs has important performance considerations. If your installation shares the RACF database with systems running releases prior to OS/390 Version 2 Release 10, it is important to use the VLF classes IRRUMAP and IRRGMAP and the UNIXMAP class to improve performance by avoiding sequential searches of the RACF database for UID and GID associations.

If your installation shares the RACF database with only systems running z/OS, or OS/390 Version 2 Release 10 or above, you may be able to achieve improved performance without using UNIXMAP and VLF. However, before you can avoid using UNIXMAP and VLF, you need to implement stage 3 of application identity mapping by running the IRRIRA00 conversion utility.

1.12 Time Sharing Option/Extended (TSO/E)

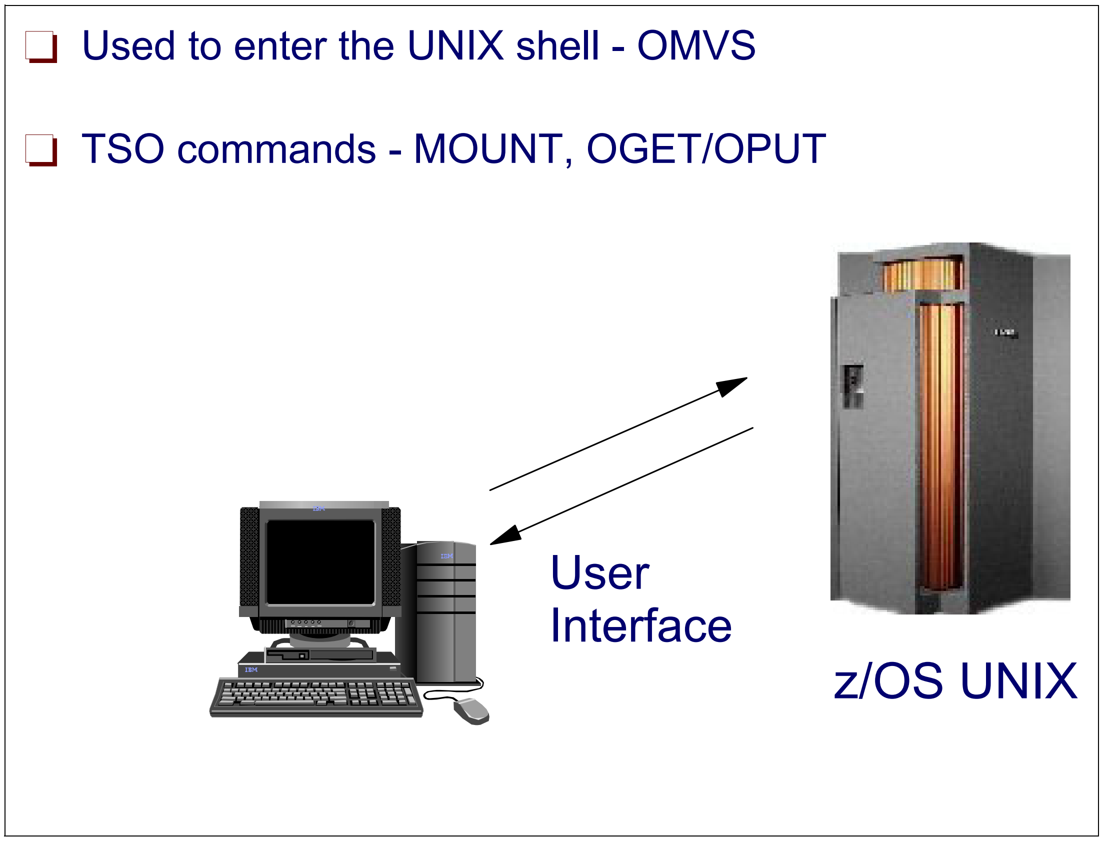

Used to enter the UNIX shell - OMVS
TSO commands - MOUNT, OGET/OPUT
User Interface
z/OS UNIX

Figure 1-12 TSO/E and z/OS UNIX

The TSO Extensions (TSO/E) licensed program is based on the Time Sharing Option (TSO), which allows users to interactively share computer time and resources.

TSO/E is composed of modules that communicate with the user and perform the work requested by the user. When a user logs on to TSO/E, the user must either specify the name of a LOGON procedure by the LOGON command or accept that user's default procedure name from the user attribute data set.

One way to enter the UNIX shell environment is by using TSO/E. A user logs on to a TSO/E session and enters the TSO/E OMVS command.

The z/OS environment has other TSO/E commands, for example, to logically mount and unmount file systems, create directories in a file system, and copy files to and from MVS 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 on how to perform these tasks using TSO/E commands, see *z/OZ UNIX System Services User's Guide,* SA22- 7801.

1.13 Workload Manager (WLM)

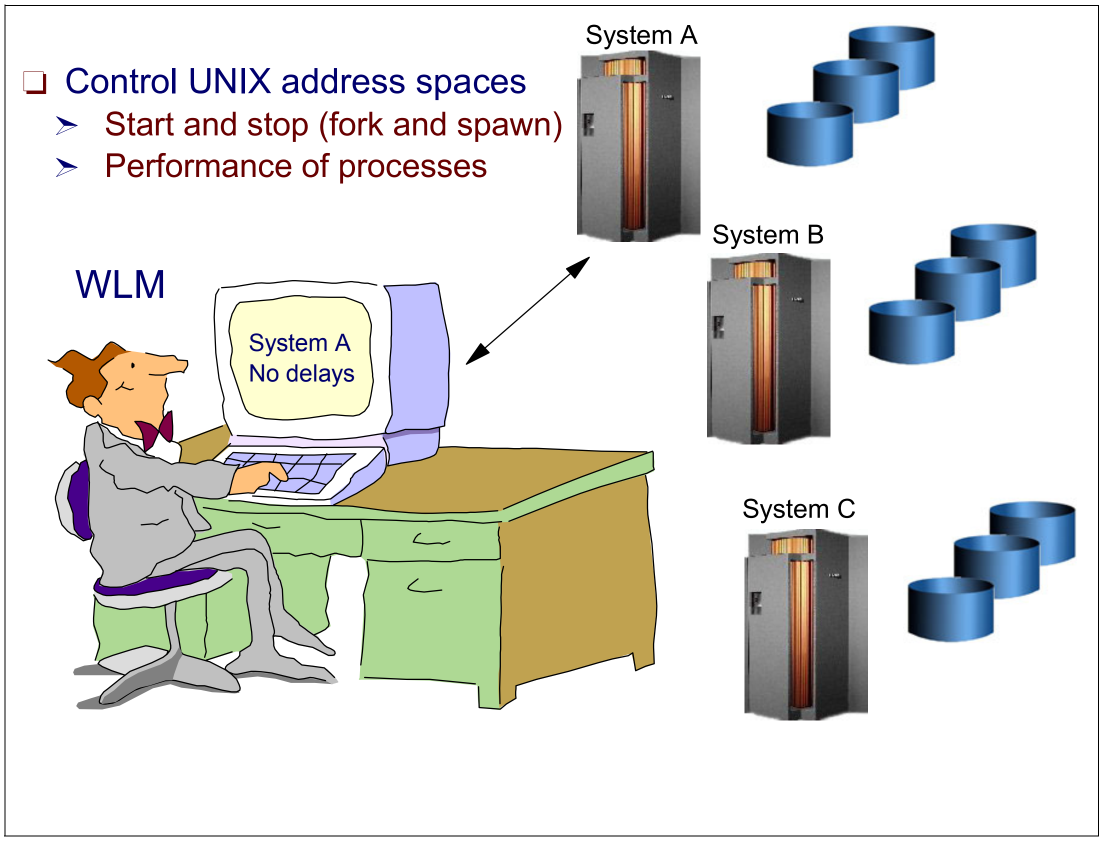


Figure 1-13 WLM and z/OS UNIX

The purpose of workload management is to balance the available system resources to meet the demands of S/390® subsystem work managers such as CICS®, BATCH, TSO, UNIX Services, and Webserver, in response to incoming work requests.

The workload manager is a component of the BCP element. When using WLM, you do not need to do any tuning or issue any commands. The kernel uses WLM to create child processes when running in goal mode or compatibility mode, or both.

Prior to OS/390 V2R4, APPC/MVS transaction initiators provided address spaces when programs issued the fork() or spawn() C function or z/OS callable services. Now, 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. Then it places the child process in a new address space. The forked address space is provided by WLM.

1.14 Tivoli® Storage Manager (TSM)

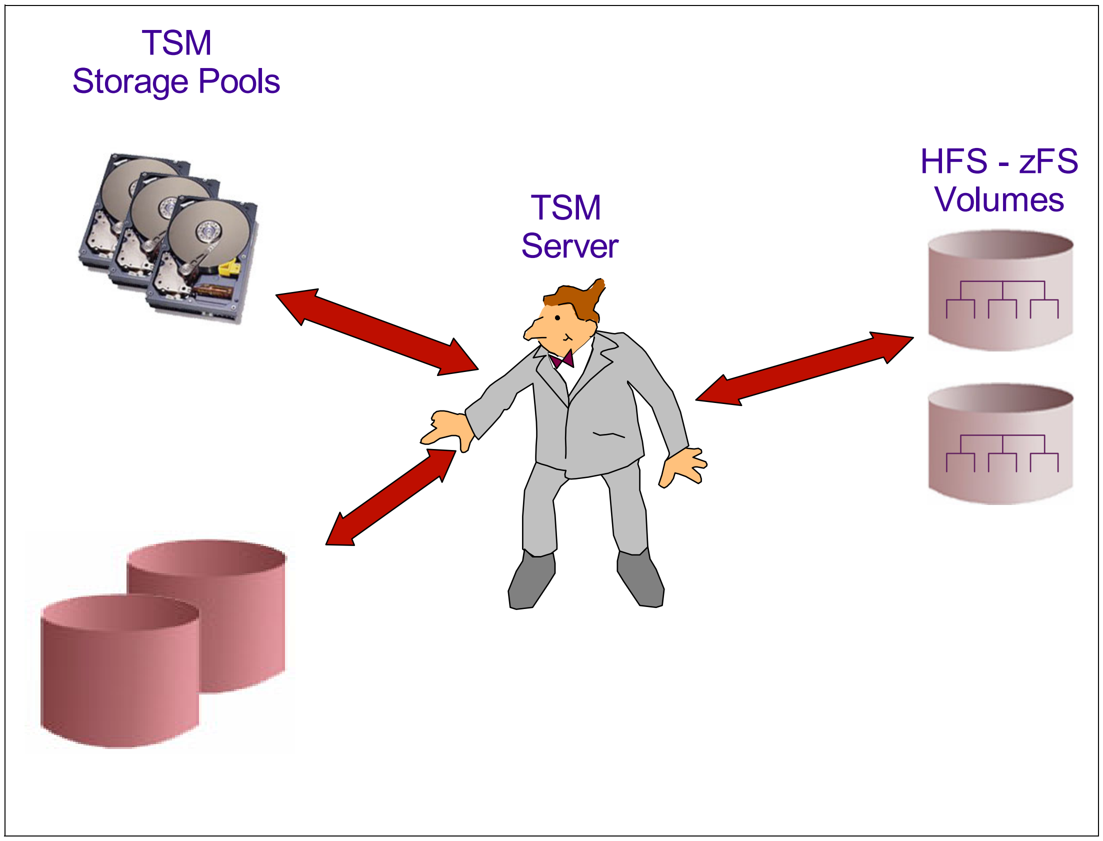


Figure 1-14 Tivoli and z/OS UNIX

Tivoli Storage Manager (TSM) is a client/server storage management product that provides administrator-controlled, highly automated, centrally scheduled, network-based backup and archive functions for workstations and LAN file servers. A TSM server backs up and/or archives data from a TSM client and stores the data in the TSM server storage pool for z/OS UNIX clients.

There are two types of backup: incremental, in which all new or changed files are backed up; and selective, in which the user backs up specific files.

Backup can be performed automatically or when the user requests it. The user can initiate a specific type of backup or start the scheduler, which will run whatever action the TSM administrator has scheduled for the user's machine.

As a TSM authorized user, you also have the authority to back up and archive all eligible files in all locally mounted file systems on your workstation, restore and retrieve all backup and archive files for your workstation from TSM storage, within the limits imposed by the UNIX file access permissions. A TSM authorized user can also grant users access to specific files in TSM storage.

Information about using the z/OS UNIX client is documented in:

- *TSM Using the Backup-Archive Clients*, SH26-4105
- *TSM Installing the Clients*, SH26-4102

Data Facility System-Managed Storage Hierarchical Storage Manager (DFSMShsm™) provides automatic backup facilities for HFS data sets. The system programmer uses DFSMShsm facilities to back up mountable file systems by backing up the HFS data sets that contain them on a regular basis; the data sets can be restored when necessary. DFSMShsm is also used for migrating (archiving) and restoring unmounted file systems.

2

z/OS UNIX overview

This chapter provides a brief overview about the most important components of UNIX System Services (z/OS UNIX). The following topics are discussed:

- The z/OS UNIX components
- Hierarchical file system structure
- z/OS UNIX interfaces
- Methods for direct login to the z/OS UNIX shell
- dbx Debugger

2.1 z/OS UNIX components

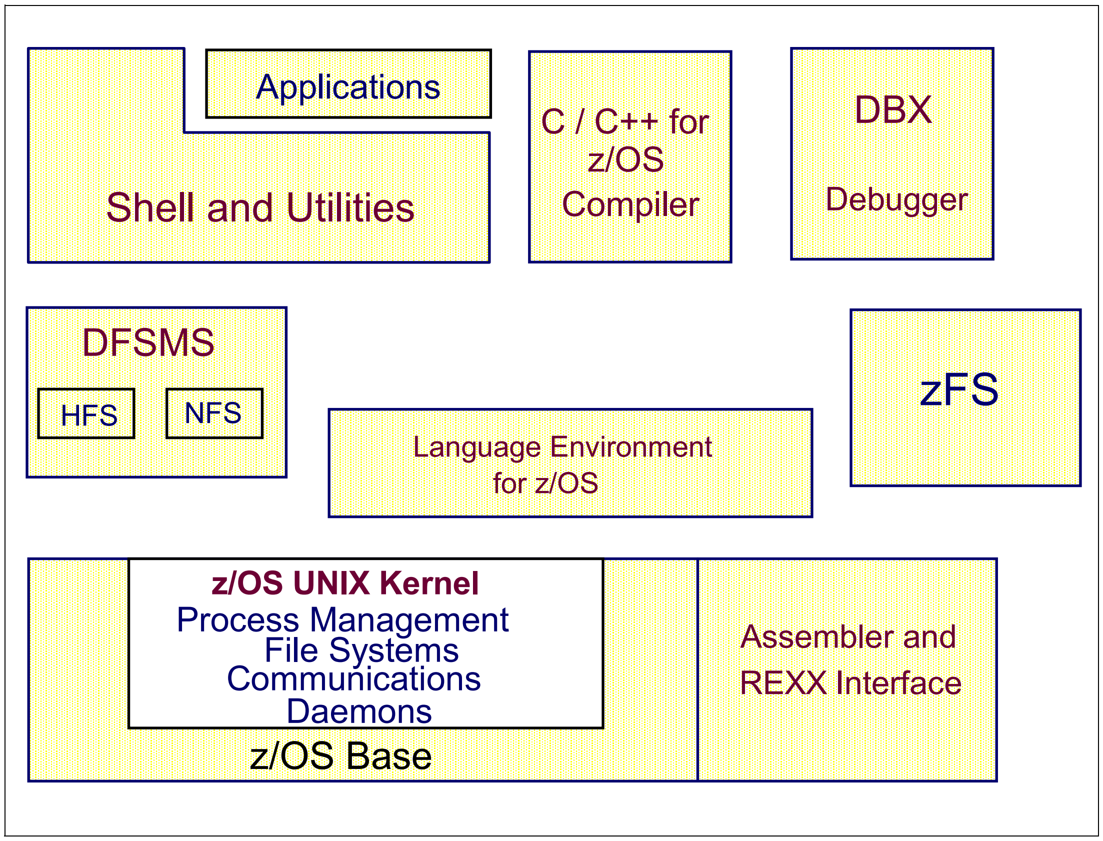


Figure 2-1 The components of z/OS UNIX

z/OS UNIX offers open interfaces for applications and interactive users on a z/OS system. The z/OS UNIX components and their functions are:

- The z/OS UNIX kernel

  At system IPL time, kernel services are started automatically. The kernel provides z/OS UNIX System Services in answer to requests from programs and the shell. The kernel manages the file system, the communications, and the program processes.

  The hierarchical file system is shipped as part of DFSMS. zFS is shipped with the Distributed File Service.

  The POSIX standard introduces a completely new terminology in the MVS environment. A typical UNIX operating system consists of a kernel which interfaces directly with the hardware. Built on the kernel is the shell and utilities layer that defines a command interface. Then there are application programs built on the shell and utilities.

  In a z/OS UNIX environment the file system is considered part of the kernel because it is allocated to the kernel. The support for the file system is provided by the DFSMS product. Figure 2-1 showing the file system as part of the kernel shows a logical view of the solution.

  The z/OS UNIX API conforms to the POSIX and XPG4 standard. OS/390 was branded as a UNIX system by the Open Group in 1996. To support the APIs, the z/OS system must provide some system services which are included in the kernel, such as the file system and communication services.

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. z/OS UNIX supplies daemons that start applications and start a user shell session when requested.

- The z/OS UNIX shell and utilities

  An interactive interface to z/OS UNIX services which interprets commands from interactive users and programs.

  The shell and utilities component can be compared to the TSO function in z/OS.

- The z/OS UNIX debugger (dbx)

  The z/OS UNIX debugger is a tool which application programmers can use to interactively debug a C program. The dbx debugger is not part of the POSIX standard. It is based on the dbx debugger that is well known in many UNIX environments.

- The C/C++ compiler and C run-time libraries

  The C/C++ compiler and C run-time libraries are needed to make executables that the kernel understands and can manage.

- DFSMS

  DFSMS manages the hierarchical file system (HFS) data sets that contain the file systems. To use kernel services in full-function mode, SMS must be active.

  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 may be independently managed.

- Language Environment (LE)

  The C compiler and Language Environment feature library are changed and extended to include support for the POSIX and XPG4 C function calls. The LE product provides a common run-time environment and language-specific run-time services for compiled programs.

  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.

- zFS

  The zSeries File System (zFS) is a “new” UNIX file system that can be used in addition to the hierarchical file system. The word new is used in quotes since this file system actually has been available since 1995 as part of the DCE component of MVS/ESA V5R2.2, OS/390, and z/OS V1R1. The file system has been known as the DCE DFS™ Local File System (LFS), sometimes referred to as Episode. It is a high performance, log-based file system. The DCE DFS Local File System is part of the OSF DFS product, and as such, it is included in the Distributed File Service base element of z/OS V1R2. zFS is a separate component of the DFS base element, so it can be serviced separately from the other components of DFS.

2.2 UNIX System Services

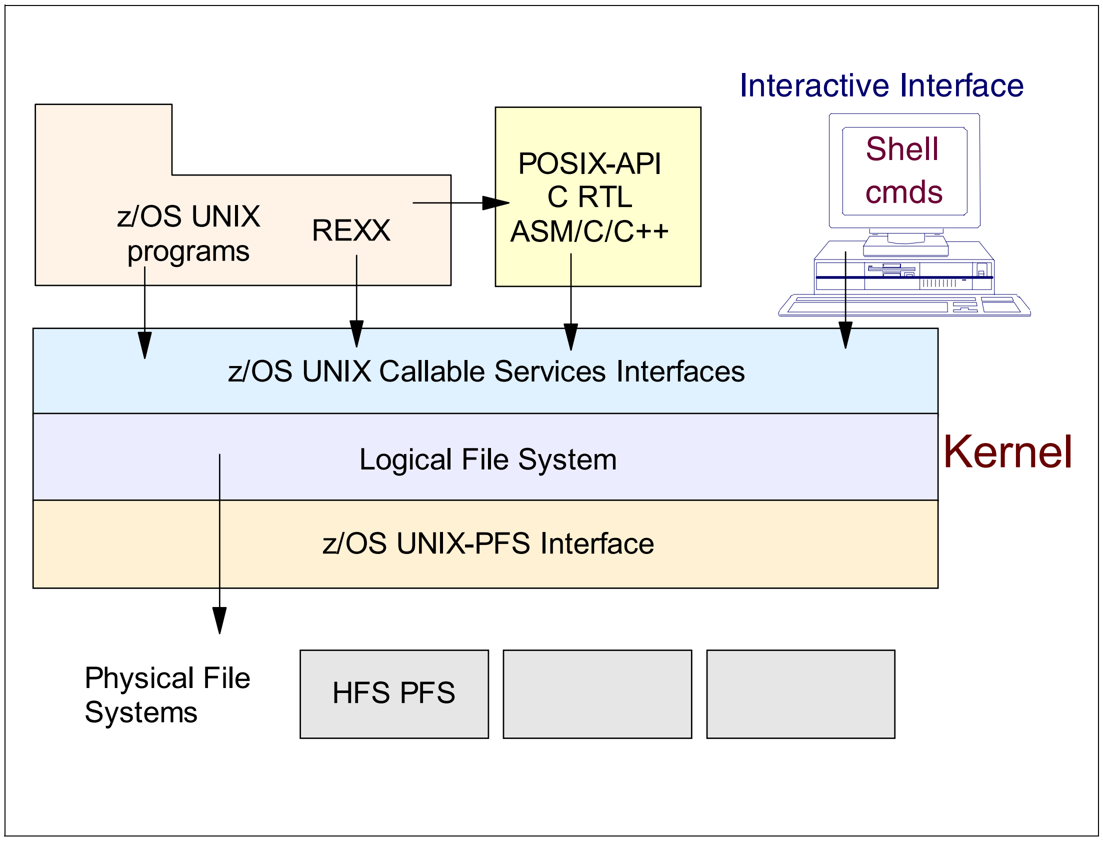


Figure 2-2 UNIX System Services and the Kernel

A user can interact with z/OS UNIX using the following interfaces:

- The Application Programming Interface (API) consists of C programming calls which can be used by C/370™ programs to access z/OS UNIX. These C calls are defined in the POSIX 1003.1 standard. Many of the C calls will use callable services to interact with the z/OS system to perform the services requested. Some C calls will interface directly with the z/OS UNIX kernel.

  The callable services can be used directly by Assembler programs to access z/OS UNIX, for example to access files in the hierarchical file system. This possibility allows other high-level languages (excluding C) and Assembler to use z/OS UNIX.

  The API interface provides the ability to run XPG4.2 programs on z/OS. A program which conforms to the XPG4.2 standard can be developed on one system and then be ported to another system, compiled and link-edited, and then executed on that system. Such a program is referred to as portable.

  A programmer can develop a program which uses a mix of standard z/OS services and z/OS UNIX. Such a program is often referred to as a mixed program. A mixed program can, for example, be a z/OS program which uses some of the Assembler callable services to access files in the hierarchical file system, or a pipe for temporary data storage.

- The interactive interface is called the z/OS UNIX shell. The shell is a command interpreter which accepts commands defined in the POSIX 1003.2 standard. Shell commands can be put together in a sequence, stored in a text file as a shell script, and then executed. The shell script is similar to z/OS CLISTs and REXX EXECs.

TSO REXX has been extended to provide access to the z/OS UNIX callable services. A REXX EXEC using UNIX System Services can be run from TSO/E, in z/OS batch, or in the shell.

You can use a set of z/OS UNIX extensions to TSO/E REXX—host commands and functions—to access kernel callable services. The z/OS UNIX extensions, called syscall commands, have names that correspond to the names of the callable services that they invoke, for example, access, chmod, and chown.

You can run a REXX program with z/OS UNIX extensions from MVS, TSO/E, the shell, or a C program. The REXX exec is not portable to an operating system that does not have z/OS UNIX installed.

The PFS interface is a set of protocols and calling interfaces between the logical file system (LFS) and the PFSs that are installed on z/OS UNIX. In a UNIX System Services environment, UNIX programs and UNIX users access their files through these interfaces. PFSs mount and unmount file systems and perform other file operations.

2.3 Physical file systems

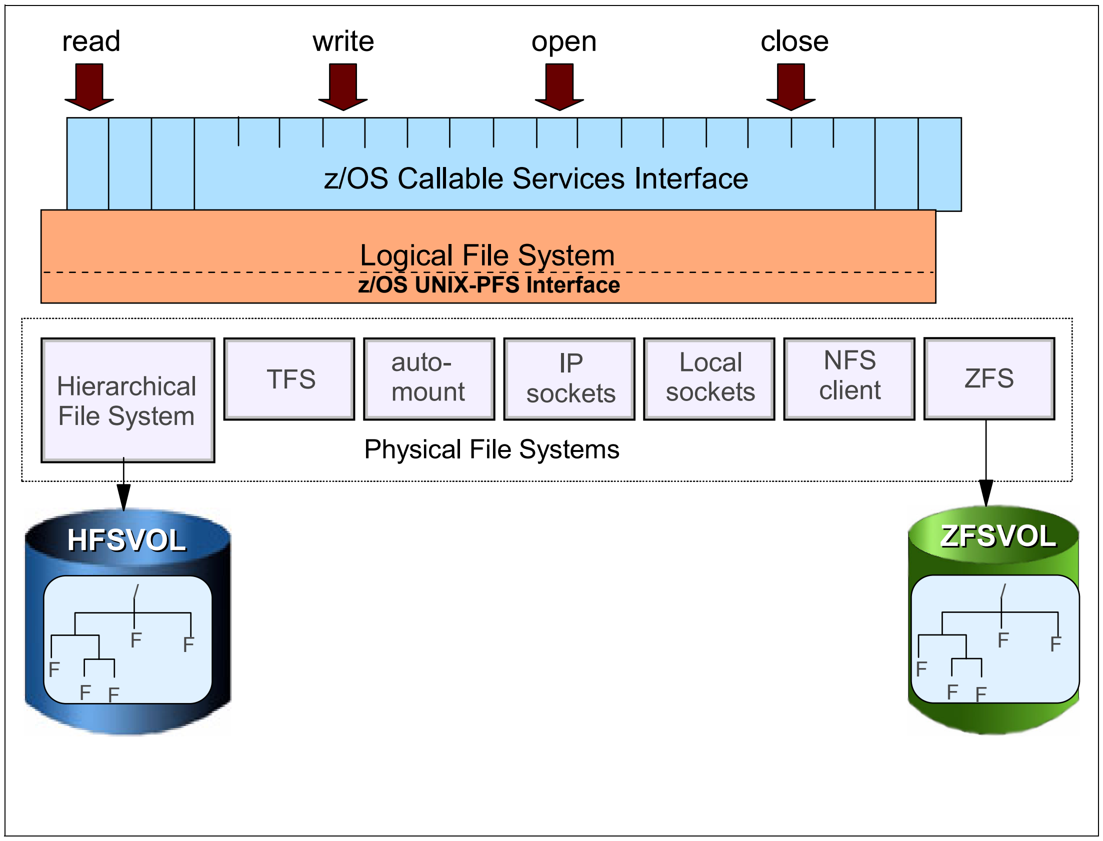


Figure 2-3 z/OS UNIX and physical file systems

A PFS controls access to data. PFSs receive and act upon requests to read and write files that they control. The format of these requests is defined by the PFS interface.

In a UNIX System Services environment, the physical file systems are defined in the BPXPRMxx PARMLIB member. zFS, as a physical file system, is also defined in the PARMLIB member. Figure 2-3 shows all the physical file systems that can be defined in a UNIX System Services environment.

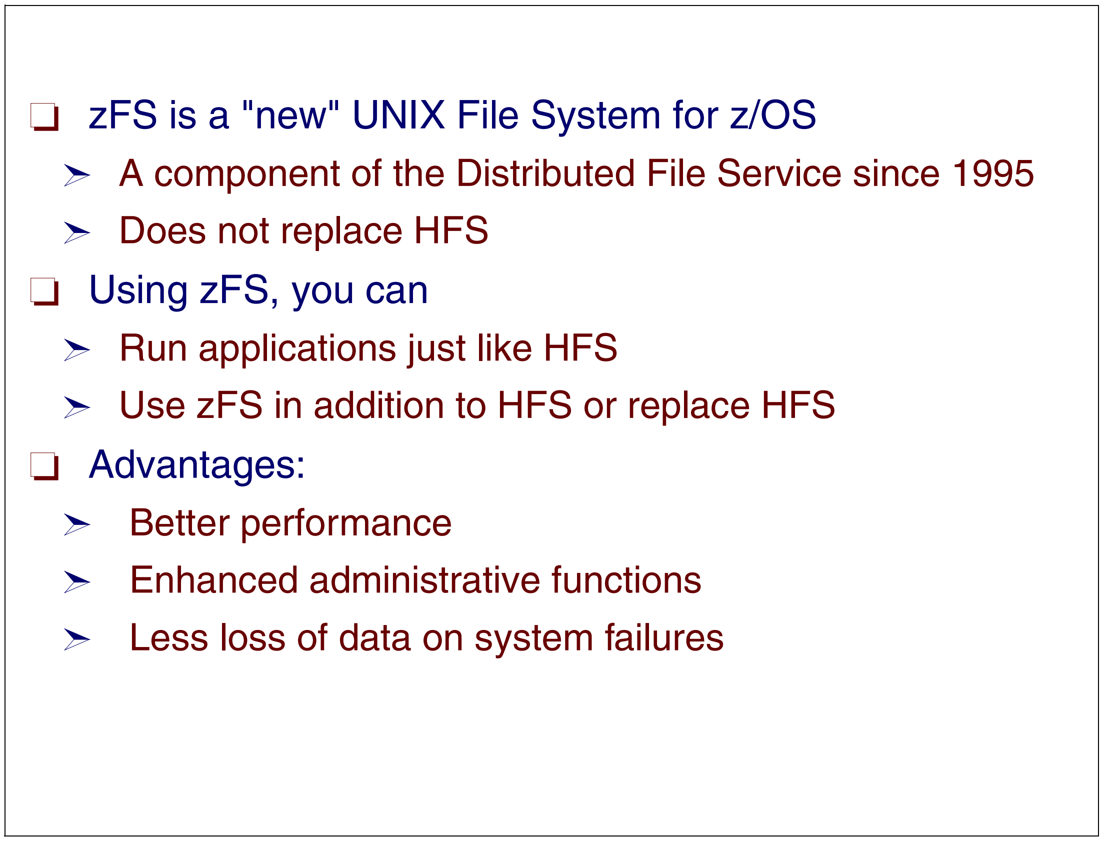
The logical file system (LFS) is called by POSIX programs, non-POSIX z/OS UNIX programs, and VFS servers.

The PFS interface is a set of protocols and calling interfaces between the LFS and the PFSs that are installed on z/OS UNIX. PFSs mount and unmount file systems and perform other file operations.

There are two types of PFSs, those that manage files and those that manage sockets:

- File management PFSs, such as HFS and zFS, deal with objects that have path names and that generally follow the semantics of POSIX files.
- Socket PFSs deal with objects that are created by the socket() and accept() functions and that follow socket semantics.

2.4 zFS file systems



- ❑ zFS is a "new" UNIX File System for z/OS
  - ➢ A component of the Distributed File Service since 1995
  - ➢ Does not replace HFS
- ❑ Using zFS, you can
  - ➢ Run applications just like HFS
  - ➢ Use zFS in addition to HFS or replace HFS
- ❑ Advantages:
  - ➢ Better performance
  - ➢ Enhanced administrative functions
  - ➢ Less loss of data on system failures

Figure 2-4 zFS file systems

The zSeries File System (zFS) is a “new” UNIX file system that can be used in addition to the hierarchical file system. The word new is used in quotes since this file system actually has been available since 1995 as part of the DCE component of MVS/ESA V5R2.2, OS/390, and z/OS V1R1. The file system has been known as the DCE DFS Local File System (LFS), sometimes referred to as Episode. It is a high performance, log-based file system. The DCE DFS Local File System is part of the OSF DFS product, and as such, it is included in the Distributed File Service base element of z/OS V1R2. zFS is a separate component of the DFS base element, so it can be serviced separately from the other components of DFS.

zFS focuses on local access. It can be locally mounted and is fully accessible from local UNIX System Services applications. It is also available remotely via the z/OS DFS SMB server from Windows® clients. zFS does not replace HFS, rather it is complementary to HFS. HFS is required for z/OS installation and the root file system must be HFS. Like HFS, zFS is a UNIX file system. It contains files and directories that can be accessed with the APIs available for HFS. In general, the application view of zFS is the same as the application view of HFS. Once a zFS file system is mounted, it is almost indistinguishable from any other mounted HFS. The benefits of using zFS are:

- Improved performance
- Underlying architecture to support additional functions
- Improved crash recovery

Initial studies indicate that there will be many environments where zFS will perform better than HFS. Initial test runs show promising results.

2.5 Hierarchical file system (HFS)

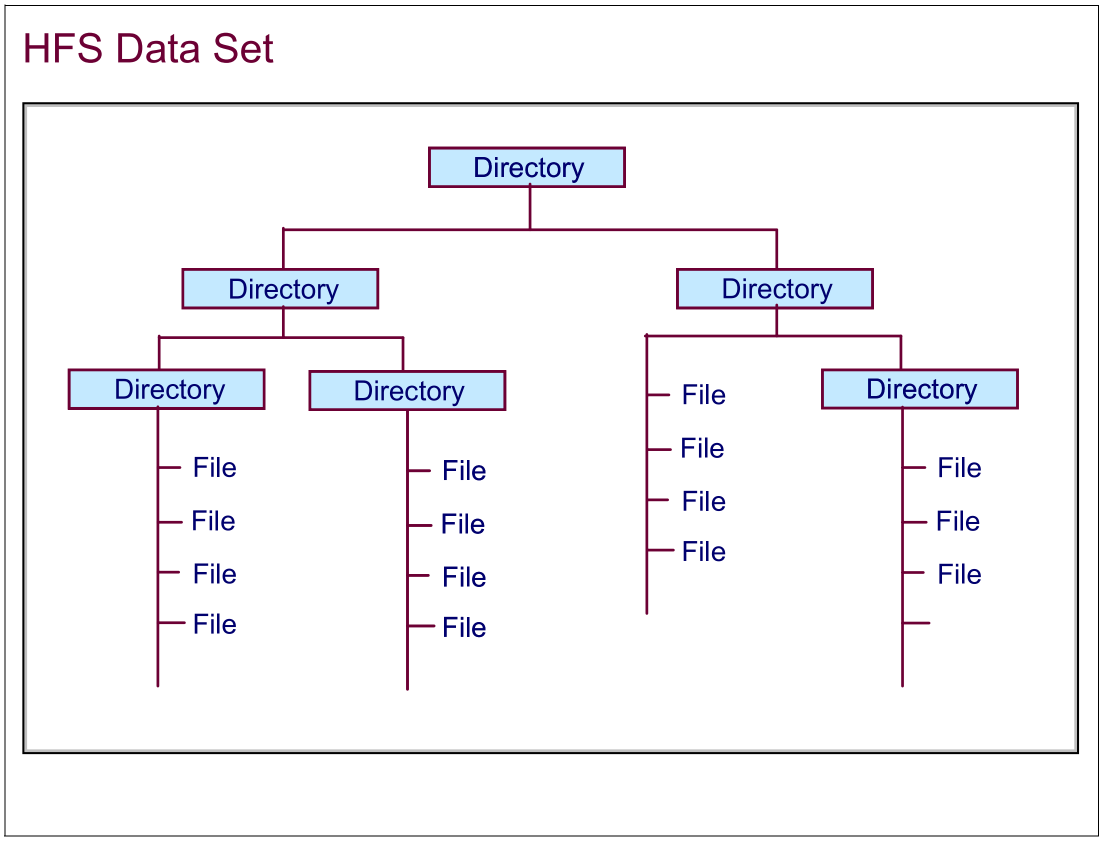


Figure 2-5 Hierarchical file system structure

A z/OS UNIX file system is hierarchical and byte-oriented. Finding a file in the file system is done by searching a directory or a series of directories. There is no concept of a z/OS catalog that points directly to a file.

A path name identifies a file and consists of directory names and a file name. A fully qualified file name, which consists of the name of each directory in the path to a file plus the file name itself, can be up to 1023 bytes long. The hierarchical file system allows for file names in mixed case.

The hierarchical file system (HFS) data set which contains the hierarchical file system is a z/OS (MVS) data set type.

The files in the hierarchical file system are sequential files, and they are accessed as byte streams. A record concept does not exist with these files other than the structure defined by an application.

The path name is constructed of individual directory names and a file name separated by the forward-slash character, for example:

```
/dir1/dir2/dir3/myfile
```

Like UNIX, z/OS UNIX is case-sensitive for file and directory names. For example, in the same directory, the file `MYFILE` is a different file than `myfile`.

HFS data sets and z/OS data sets can reside on the same DASD volume.

The integration of the HFS file system with existing MVS file system management services provides automated file system management capabilities which may not be available on other POSIX platforms. This allows file owners to spend less time on tasks such as backup and restore of entire file systems.

2.6 File system data sets

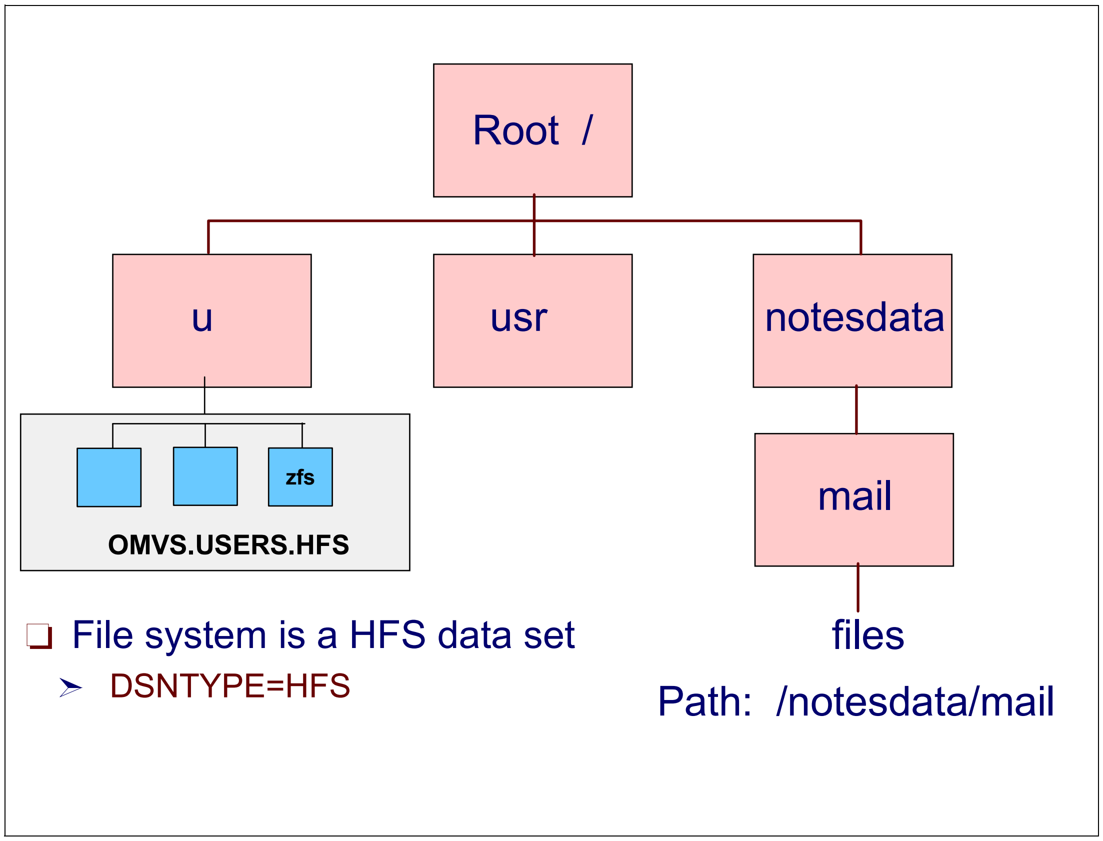


Figure 2-6 File system data sets

z/OS UNIX files are organized in a hierarchy, as in a UNIX system. All files are members of a directory, and each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the root directory.

MVS views an entire file hierarchy as a collection of hierarchical file system data sets (HFS data sets). Each HFS data set is a mountable file system. DFSMS facilities can be used to manage an HFS data set, and DFSMS Hierarchical Storage Manager (DFSMShsm*) is used to back up and restore an HFS data set.

The root file system is the first file system mounted. Subsequent file systems can be mounted on any directory within the root file system or on a directory within any mounted file system.

A file in the hierarchical file system is called an HFS file. HFS files are byte-oriented, rather than record-oriented, as are MVS data sets.

2.7 z/OS UNIX programs (processes)

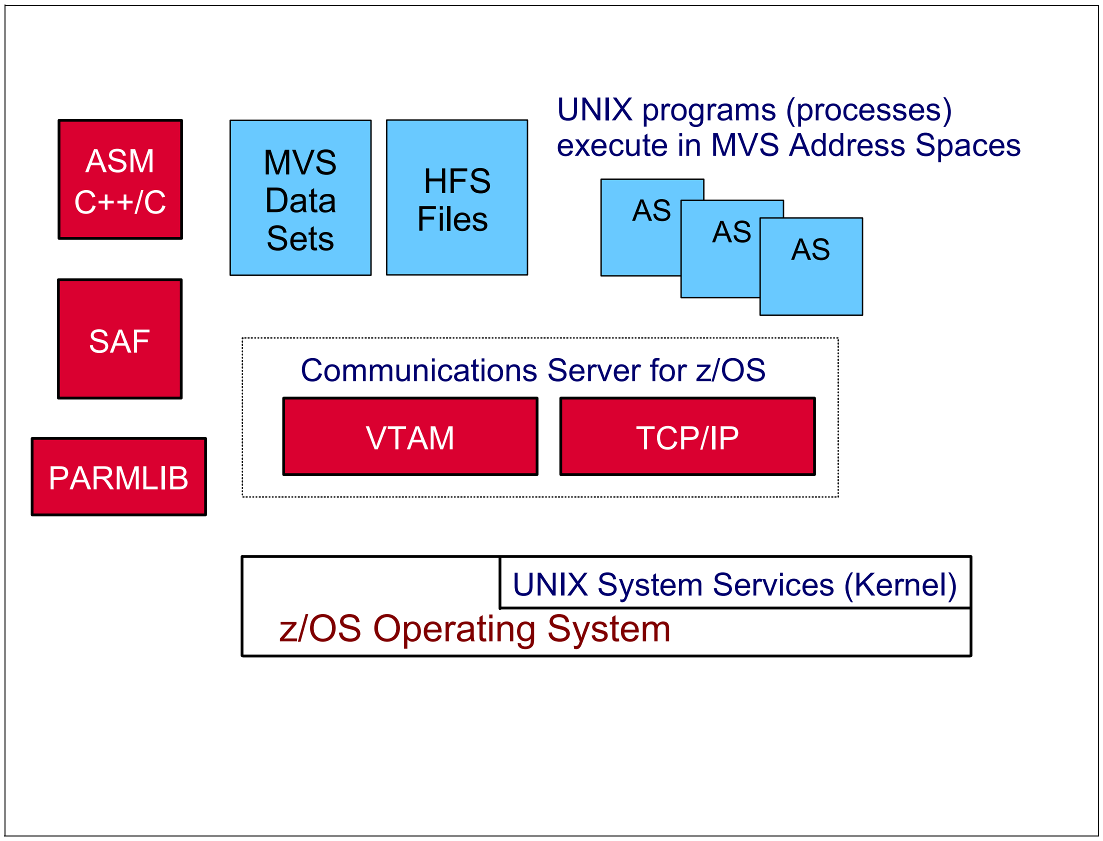

ASM
C++/C
SAF
PARMLIB
MVS Data Sets
HFS Files
UNIX programs (processes) execute in MVS Address Spaces
AS
AS
AS
Communications Server for z/OS
VTAM
TCP/IP
UNIX System Services (Kernel)
z/OS Operating System

Figure 2-7 z/OS UNIX programs (processes) and components

A process is a program using kernel services. The program can be created by a fork() function, fork callable service, or spawn() function; or the program can be dubbed because it requested kernel services. The three types of processes are:

- User processes, which are associated with a program or a shell user
- Daemon processes, which perform continuous or periodic system-wide functions, such as printer spooling
- Kernel processes, which perform system-wide functions for the kernel such as cleaning up zombie processes (init process)

When you enter a shell command, you start a process that runs in an MVS address space. When you enter that command, the z/OS shell runs it in its own process group. As such, it is considered a separate job and the shell assigns it a job identifier—a small number known only to the shell. A shell job identifier identifies a shell job, not an MVS job. When the process completes, the system displays the shell prompt.

UNIX programs running in MVS address spaces use or access the following:

ASM/C++/C	These programming languages can be used to create the programs.
SAF	A security product is required when running UNIX System Services. SAF is the interface to RACF, Top Secret, or ACF2.
BPXPRMxx	This parmlib member determines the number of processes that may be started in the z/OS system.
MVS data sets	UNIX programs may access MVS data sets.

HFS files	UNIX programs may access HFS files.
TCP/IP	Workstation users can enter the shell environment by using **`rlogin`** or **`telnet`** in a TCP/IP network. User-written applications can use TCP/IP as a communication vehicle.
VTAM	Workstation users can access TSO/E through VTAM. z/OS UNIX is then accessed from TSO/E.

2.8 Create a process

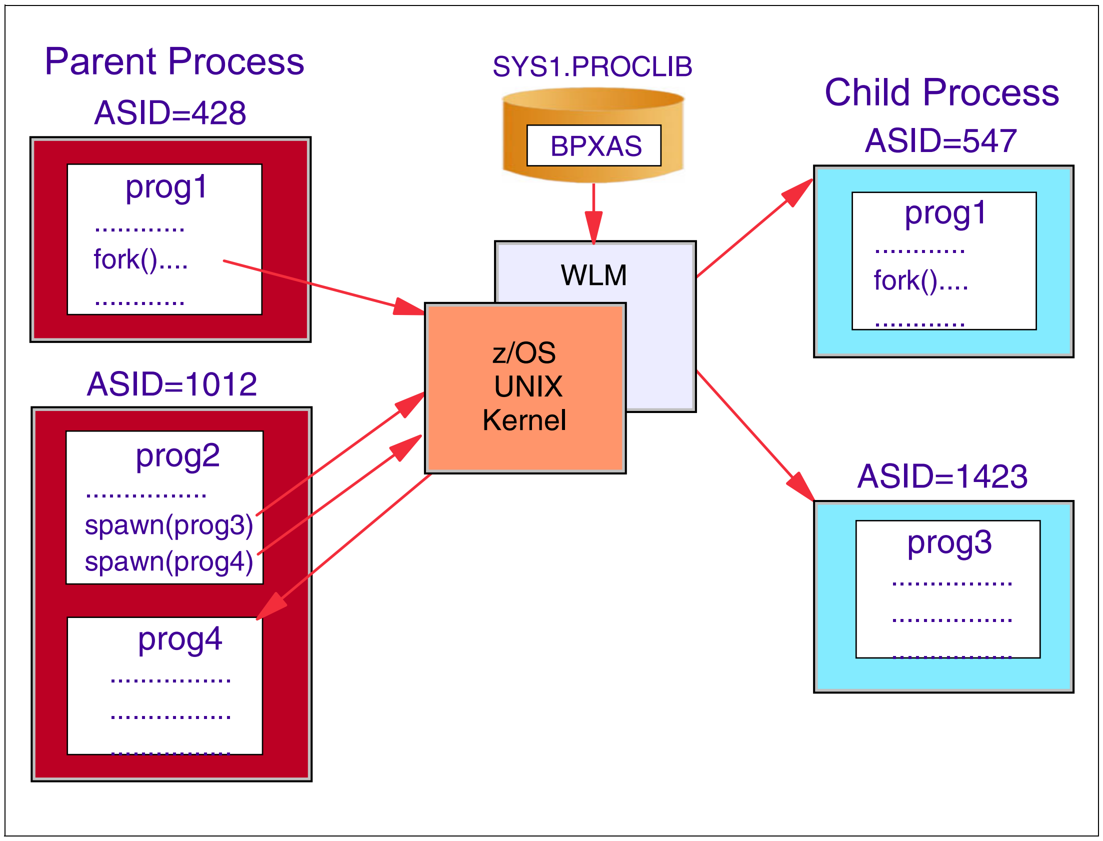


Figure 2-8 Creating a process

Fork() is a POSIX/XPG4 function which creates a duplicate process referred to as a child process. The process that issues the fork() is referred to as the parent process.

With z/OS UNIX, a program that issues a fork() function creates a new address space that is a copy of the address space where the program is running. The fork() function does a program call to the kernel, which uses WLM/MVS facilities to create the child process. The contents of the parent address space are then copied to the child address space.

After the fork() function, the program in the child AS will start at the same instruction as the program in the parent AS. Control is returned to both programs at the same point. The only difference that the program sees is the return code from the fork() function. A return code of zero is returned to the child after a successful fork(). All other return codes are valid only for the parent.

Once the child address space has been created, the child gets the required storage from a STORAGE request. The kernel then copies the contents of the parent AS to the child AS using the MVCL instruction. Once the copy has been completed, a short conversation between the kernel and the child process takes place. At this point, both the parent and child process are activated. The program in the child AS gets control at the same point as the program in the parent AS. The only difference is the return code from the fork() function.

The child address space is almost an identical copy of the parent address space. User data, for example private subpools, and system data, like RTM (Recovery Termination Management) control blocks, are identical.

A process will be created by using any of the following methods:

- **C fork() function:** Creates a child process which is identical to the parent process in a new address space scheduled by the Workload Manager (WLM).
- **C spawn() function:** Creates a child process which will execute a different program than the parent, either in a new address space scheduled by WLM, or in the same address space as the parent process (local spawn).
- **z/OS UNIX callable services:** When a program uses a z/OS UNIX assembler callable service, the z/OS address space will be *dubbed* a z/OS UNIX process. The address space will get a PID. The dub will not result in a new address space.

The kernel interfaces to WLM to create the new address space for a fork or spawn. WLM uses an IBM-supplied procedure **BPXAS** to start up a new address space. This new address space will then initialize the UNIX child process to run in the address space. After the child process completes, this address space can be reused for another fork or spawn. If none is waiting, BPXAS will time out after being idle for 30 minutes.

2.9 z/OS UNIX processes

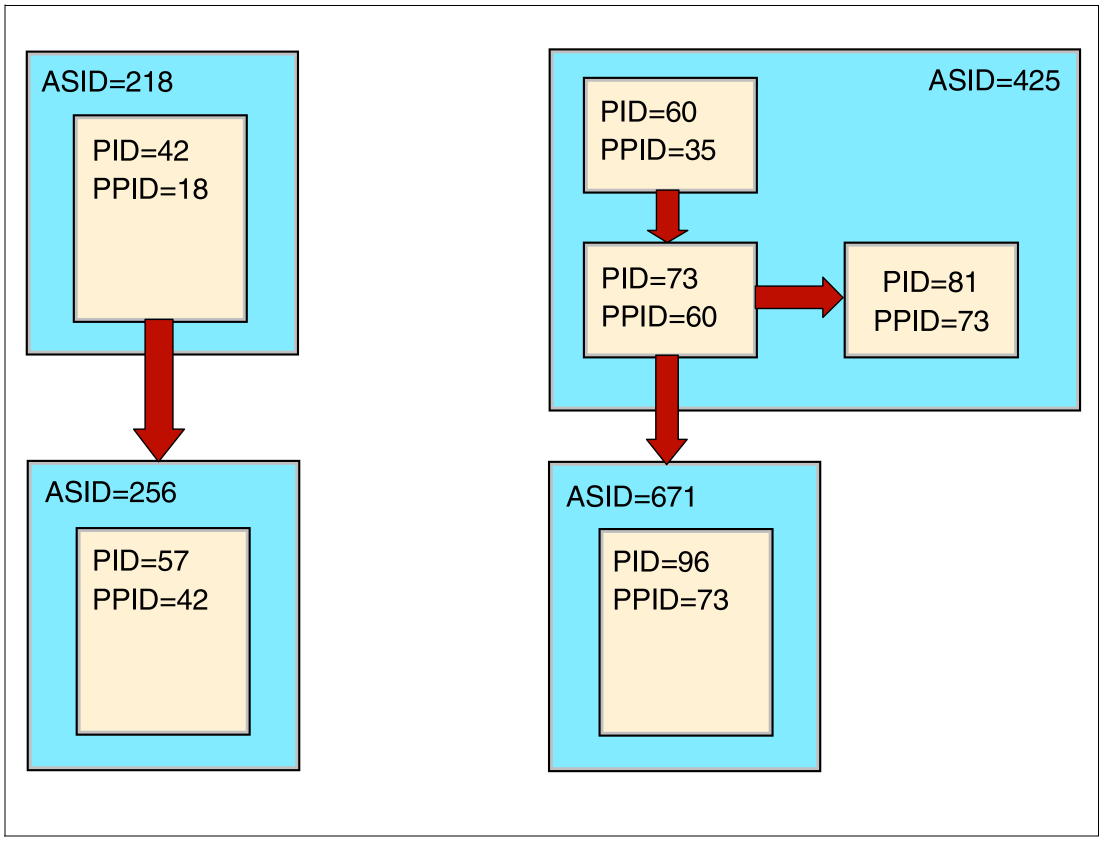


Figure 2-9 z/OS UNIX processes

z/OS UNIX uses processes to run programs, and to associate resources to the programs. A z/OS address space can contain one or multiple processes. A process is created by another process, or by a request for a z/OS UNIX service. The process that creates a new process is called a parent process and the new process is called a child process. There will be a hierarchy of processes in the system.

Every process is identified by a process id (PID) and is associated with its parent process by a parent process id (PPID).

z/OS UNIX supports processes that run in unique address spaces. These would be created by fork() and exec() services. It also supports local processes. These will share an address space and are created by the spawn() service.

The system also assigns a process group identifier (PGID) and a process identifier (PID). When only one command is entered, the PGID is the same as the PID. The PGID can be thought of as a system-wide identifier. If you enter more than one command at a time using a pipe, several processes, each with its own PID, will be started. However, these processes all have the same PGID and shell job identifier. The PGID is the same as the PID for the first process in the pipe. Process identifiers associated with a process are as follows:

PID A process ID. A unique identifier assigned to a process while it runs. When the process ends, its PID is returned to the system. Each time you run a process, it has a different PID (it takes a long time for a PID to be reused by the system). You can use the PID to track the status of a process with the `ps` command or the `jobs` command, or to end a process with the `kill` command.

PGID Each process in a process group shares a process group ID (PGID), which is the same as the PID of the first process in the process group. This ID is used for signaling related processes. If a command starts just one process, its PID and PGID are the same.

PPID A process that creates a new process is called a parent process; the new process is called a child process. The parent process ID (PPID) becomes associated with the new child process when it is created. The PPID is not used for job control.

A process can create one or more child processes, and the child processes can be parent processes of new child processes, thus creating a hierarchy of related processes. The PPID maintains the relationship between processes. Usually, a process creates a child process to perform a separate task, for example a shell command. The child process ends when the task is completed while the parent process continues to execute. If for some reason a parent process should terminate before a child process, the child process will become an orphan process. Orphan processes are inherited by the first process created in the system, called the *init* process.

2.10 MVS data sets versus file system files

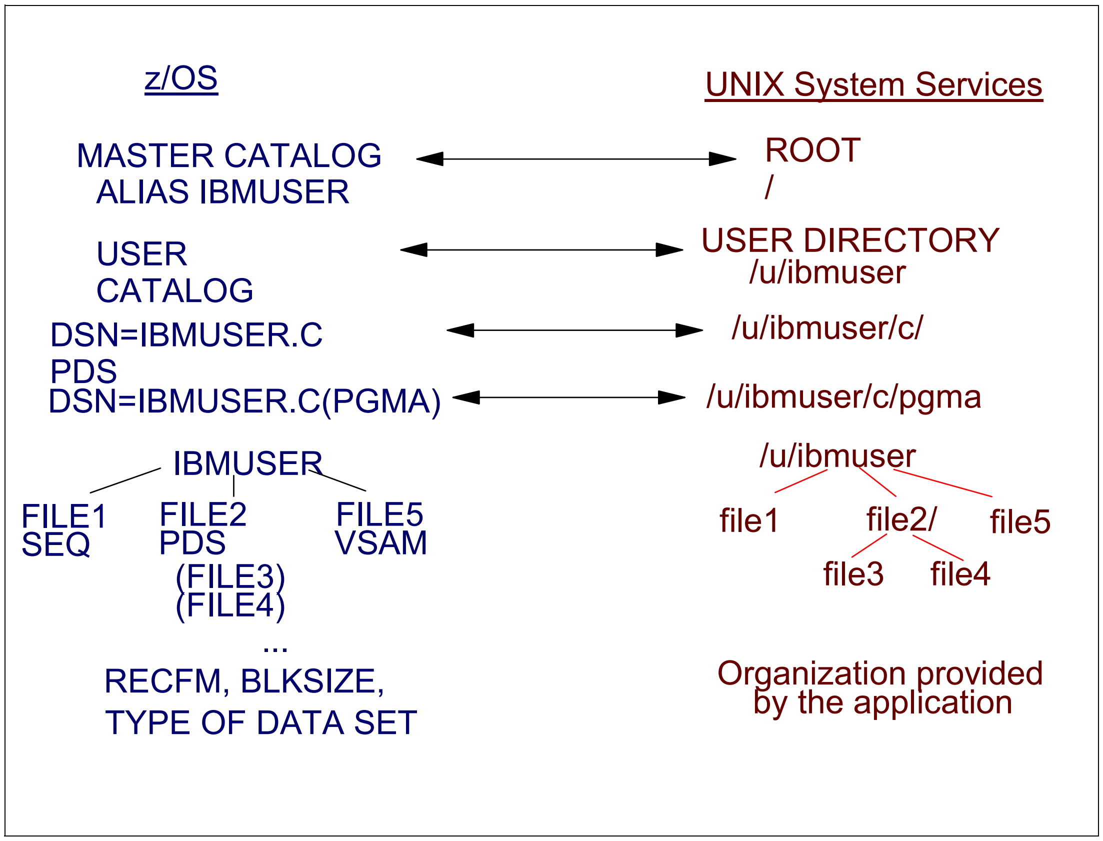


Figure 2-10 Comparison of MVS data set and file system files

The z/OS master catalog is analogous to the root directory in a hierarchical file system.

The user prefix assigned to MVS data sets points to a user catalog. The organization of the user catalog is analogous to a user directory (/u/ibmuser) in the file system. Typically, one user owns all the data sets whose names begin with his user prefix. For example, the data sets belonging to the TSO/E user ID IBMUSER all begin with the prefix IBMUSER. There could be data sets named IBMUSER.C, and IBMUSER.C(PGMA).

In the file system, ibmuser would have a user directory named /u/ibmuser. Under that directory there could be subdirectories named /u/ibmuser and /u/ibmuser/c/pgma.

Of the various types of MVS data sets, a partitioned data set (PDS) is most like a user directory in the file system. In a partitioned data set such as IBMUSER.C, you could have members PGMA, PGMB, and so on. For example, you could have IBMUSER.C(PGMA) and IBMUSER.C(PGMB). A subdirectory such as /u/ibmuser/c can hold many files, such as pgma, pgmb, and so on.

All data written to the hierarchical file system can be read by all programs as soon as it is written. Data is written to a disk when a program issues an fsync().

2.11 HFS data sets

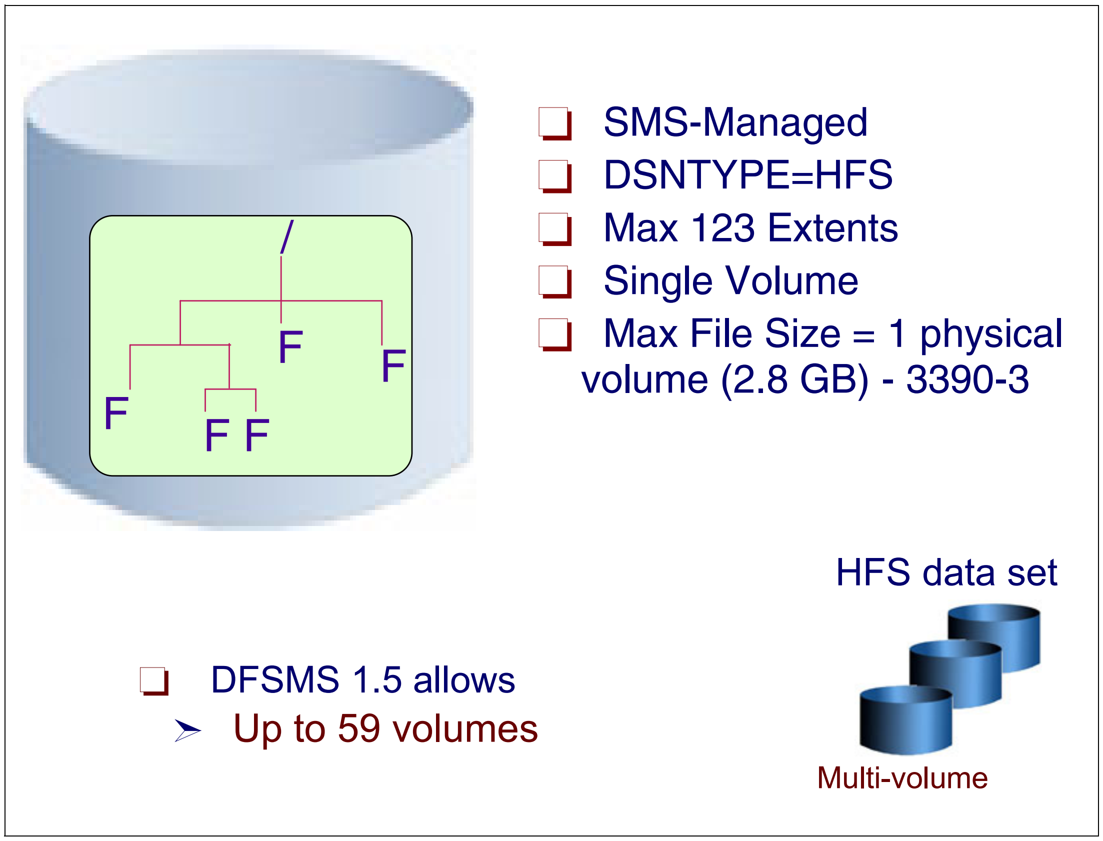


Figure 2-11 HFS data sets

A z/OS UNIX hierarchical file system is contained in a data set type called HFS. An HFS data set must reside on an SMS-managed volume, and it is a single volume data set. HFS data sets can reside with other MVS data sets on SMS-managed volumes or non-SMS-managed volumes. Multiple systems can share an HFS data set if it is mounted in read-only mode. Beginning with OS/390 V2R9, sharing can be done in read-write mode.

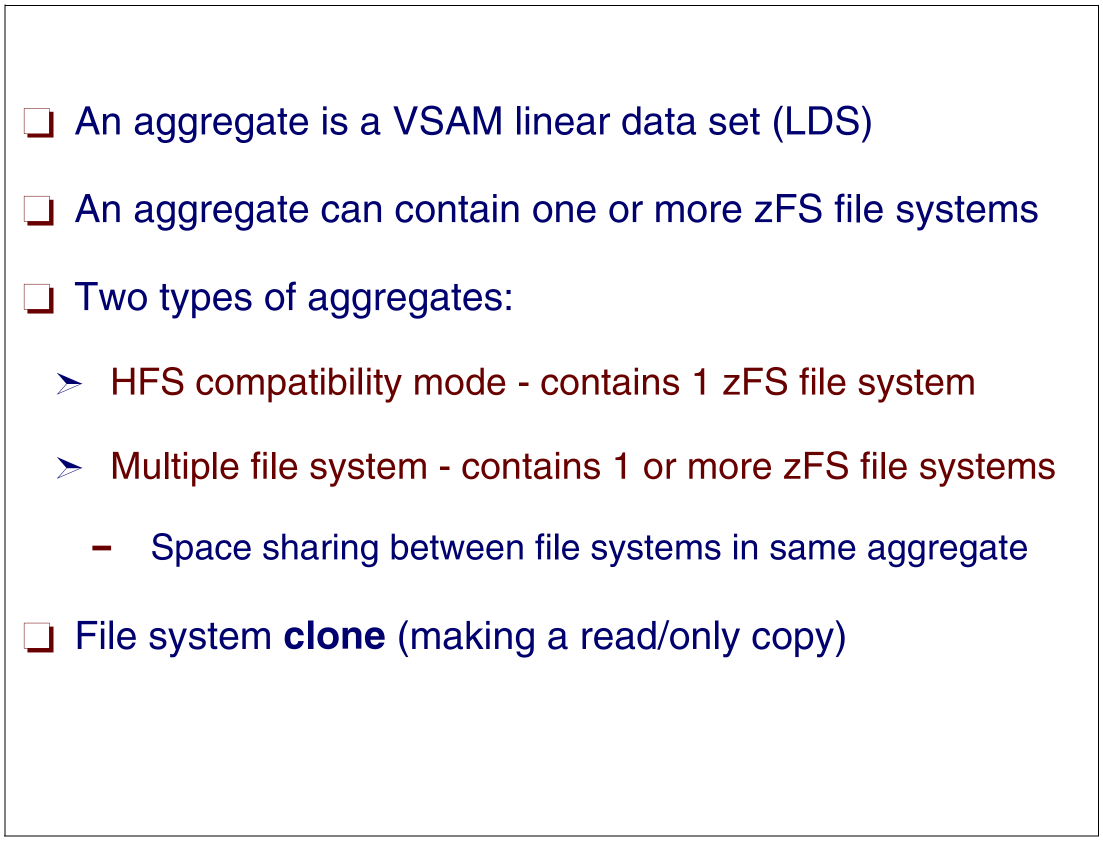
Note: APAR OW35441 now gives you the ability to allocate PDSE and HFS data sets on unmanaged (non-SMS) volumes, if running DFSMS 1.4 or DFSMS 1.5.

An HFS data set can have up to 123 extents, and the maximum size of the data set is one physical volume. For a 3390-Model 3 the maximum size is 2.838 GB. HFS data sets can only be accessed by z/OS UNIX.

An HFS data set is allocated by specifying HFS in the DSNTYPE parameter. You can also define a data class for HFS data sets. All HFS data sets must be system-managed, and an individual HFS data set can reside on only one DASD volume. OS/390 V2R7 began to support multivolume access up to 59 physical volumes. For the 3390-Model 3 DASD the maximum size was 2.8 GB (59 * 2.8 GB = 165.2 GB).

The z/OS UNIX file system is composed of multiple HFS data sets connected to each other in a hierarchy. On top of the hierarchy is the root file system. The root file system contains system directories and files. Most installations will have user data in separate HFS data sets which are connected to the root. Connecting a file system to another is called mounting.

2.12 zFS aggregates



- An aggregate is a VSAM linear data set (LDS)
- An aggregate can contain one or more zFS file systems
- Two types of aggregates:
  - HFS compatibility mode - contains 1 zFS file system
  - Multiple file system - contains 1 or more zFS file systems
    - Space sharing between file systems in same aggregate
- File system **clone** (making a read/only copy)

Figure 2-12 zFS aggregates

A zFS aggregate is a data set that contains zFS file systems. The aggregate is a VSAM Linear Data Set (VSAM LDS) and is a container that can contain one or more zFS file systems. An aggregate can only have one VSAM LDS, but it can contain an unlimited number of file systems. The name of the aggregate is the same as the VSAM LDS name.

Sufficient space must be available on the volume or volumes, as multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during any extension of a primary allocation. VSAM LDSs greater than 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set.

After the aggregate is created, formatting of the aggregate is necessary before any file systems can exist in it. A zFS file system is a named entity that resides in a zFS aggregate. It contains a root directory and can be mounted into the UNIX System Services hierarchy. While the term file system is not a new term, a zFS file system resides in a zFS aggregate, which is different from an HFS file system. zFS aggregates come in two types:

- Compatibility mode aggregates
- Multi-file system aggregates

zFS allows an administrator to make a read-only clone of a file system in the same aggregate. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata is copied.

2.13 zFS compatibility mode aggregate

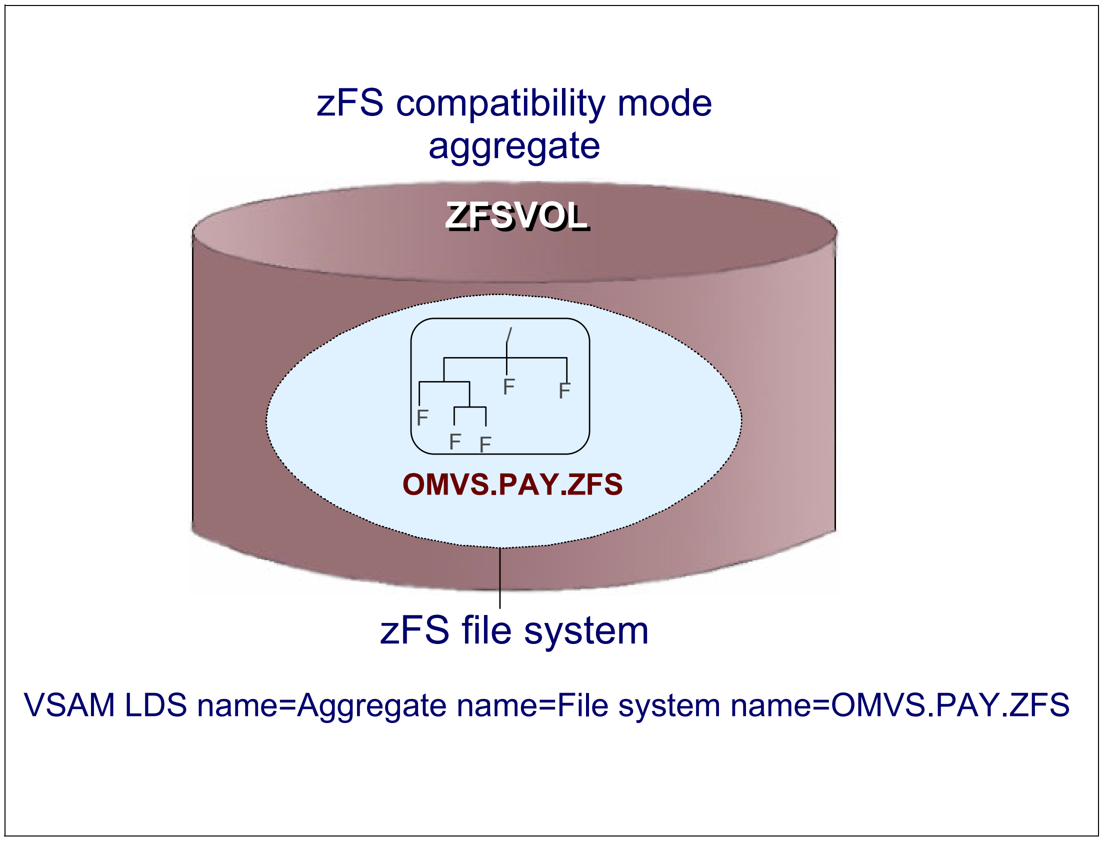

zFS compatibility mode aggregate
ZFSVOL
/
F
F
F
F
F
OMVS.PAY.ZFS
zFS file system
VSAM LDS name=Aggregate name=File system name=OMVS.PAY.ZFS

Figure 2-13 zFS compatibility mode aggregate

A compatibility mode aggregate can contain only one zFS file system, making this type of aggregate more like an HFS file system. This is flagged in the aggregate when it is created. The name of the file system is the same as the name of the aggregate, which is the same as the VSAM LDS cluster name. The file system size (called a quota) in a compatibility mode aggregate is set to the size of the aggregate. Compatibility mode aggregates are more like an HFS data set, except that they are VSAM linear data sets instead of HFS data sets. We recommend that you start using compatibility mode aggregates first, since they are more like the familiar HFS data sets.

2.14 Multiple file mode aggregate

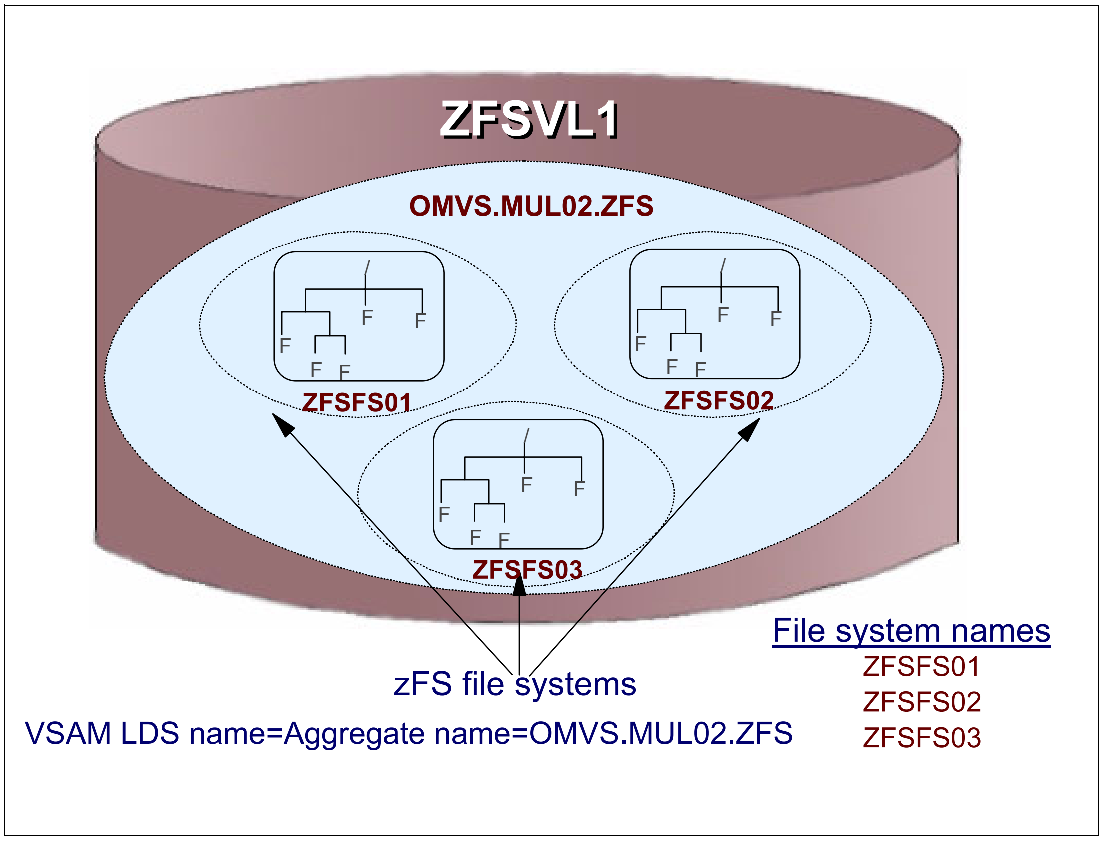


Figure 2-14 zFS multiple mode aggregate

A multi-file mode aggregate allows the administrator to define multiple zFS file systems in a single aggregate. This type of aggregate can contain one or more zFS file systems. This allows space that becomes available when files are deleted in one file system to be made available to other file systems in the same aggregate data set or space sharing.

Space sharing

Space sharing means that if you have multiple file systems in a single data set, and files are removed from one of the file systems—which frees DASD space—another file system can use that space when new files are created. This new type of file system is called a multi-file system aggregate.

The multiple file system aggregate OMVS.MUL02.ZFS, shown in the figure, can contain multiple zFS file systems. This makes it possible to do space sharing between the zFS file systems within the aggregate.

The multiple file system aggregate has its own name. This name is assigned when the aggregate is created. It is always the same as the VSAM LDS cluster name. Each zFS file system in the aggregate has its own file system name. This name is assigned when the particular file system in the aggregate is created. Each zFS file system also has a predefined maximum size, called the quota.

2.15 zFS file system clone

- zFS file system clone is a "copy" of a zFS file system
  - In the same aggregate
  - The clone file system is R/O
  - Only the metadata is copied
  - The cloned file system is called *filename*.bak

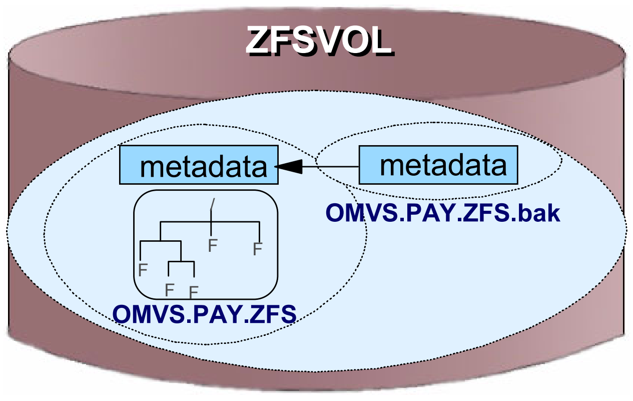


Figure 2-15 zFS file system clone

You can make a clone of a zFS file system. The zFS file system, which is the source file system for the clone, is referred to as the read-write file system. The zFS file system that is the result of the clone operation is called the backup file system. The backup file system is a read-only file system and can only be mounted as read-only. You can create a backup file system for every read-write file system that exists.

The aggregate containing the read-write file system to be cloned must be attached. Creating a file system clone is accomplished with the **`zfsadm clone`** command, as follows:

```
zfsadm clone -filesystem OMVS.CMP01.ZFS
```

Look for the following successful clone message:

```
IOEZ00225I File system OMVS.CMP01.ZFS successfully cloned.
```

When a file system is cloned, a copy of the file system is created in the same aggregate; space must be available in the aggregate for the clone to be successful. The file system name of the backup file system is the same as the original (read-write) file system with .bak (in lower case) appended to the file system name, as shown in the visual. This means that you need to limit the length of a file system name to 40 characters if you want to clone it.

2.16 z/OS UNIX interactive interfaces

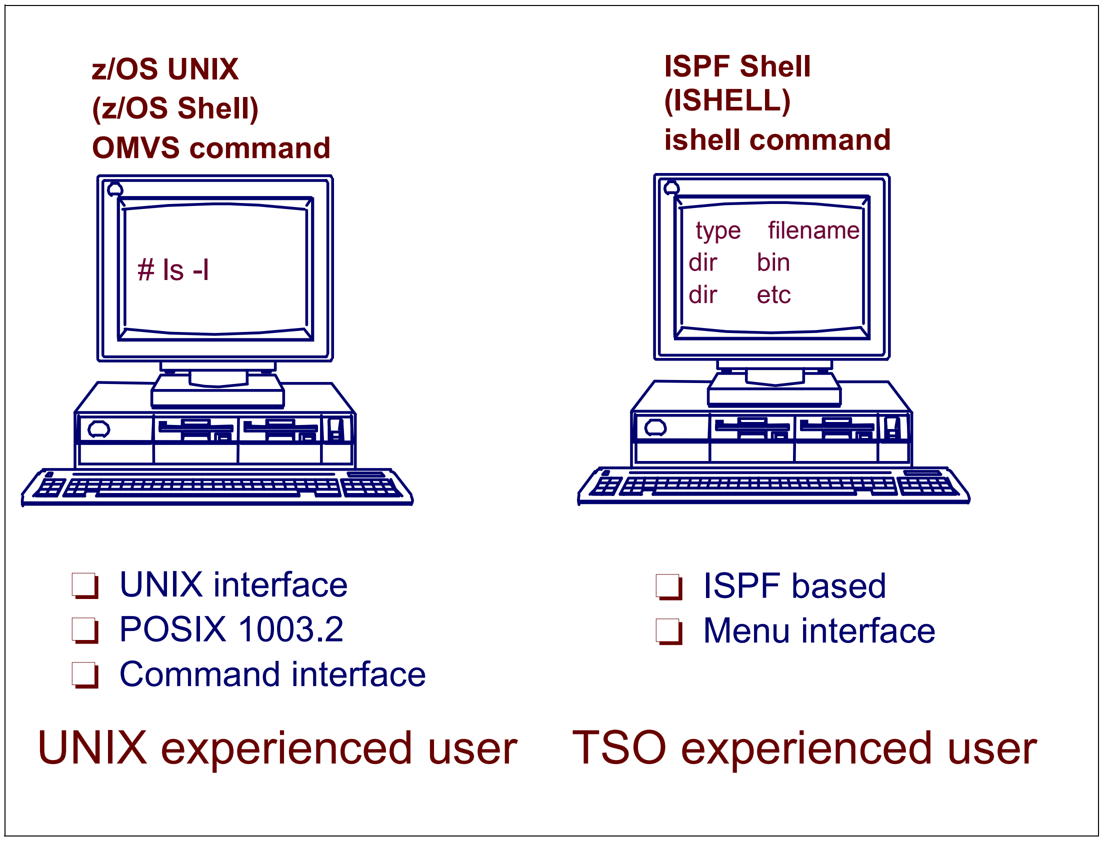


Figure 2-16 z/OS UNIX interactive interfaces

Figure 2-16 is an overview of the two interactive interfaces, z/OS UNIX shell and the ISHELL. In addition, there are some TSO/E commands to support z/OS UNIX, but they are limited to certain functions such as copying files and creating directories.

The z/OS UNIX shell is based on the UNIX System V shell and has some of the features from the UNIX Korn shell. The POSIX standard distinguishes between a *command*, which is a directive to the shell to perform a specific task, and a *utility*, which is the name of a program callable by name from the shell. To the user, there is no difference between a command and a utility.

Interactive users of z/OS UNIX have a choice between using a UNIX-like interface (the shell), a TSO interface (TSO commands), and an ISPF interface (ISPF CUA® dialog). With these choices, users can choose the interface which they are most familiar with and get a quicker start on z/OS UNIX.

The z/OS UNIX shell provides the environment that has the most functions and capabilities. Shell commands can easily be combined in pipes or shell scripts and thereby become powerful new functions. A sequence of shell commands can be stored in a text file which can be executed. This is called a shell script. The shell supports many of the features of a regular programming language.

There are some TSO commands which provide support for UNIX System Services, as follows:

ISHELL The ISHELL command will invoke the ISPF shell. The ISHELL is a good starting point for users familiar with TSO and ISPF who wantor need to use z/OS UNIX. The ISHELL provides CUA panels where users can work with the hierarchical file system. There are also panels for mounting/unmounting file systems and for doing some z/OS UNIX administration.

OMVS The OMVS command used to invoke the z/OS UNIX shell.

The ISHELL is an ISPF dialog for users and system administrators which can be used instead of shell commands to perform many tasks related to file systems, files, and z/OS UNIX user administration.

The REXX support for z/OS UNIX is not really an interactive interface, but we chose to introduce it here since it is most often used in TSO or in the shell. The SYSCALL environment is not built into TSO/E, but an external function call called SYSCALLS will initialize the environment. Note that the shell is the initial host environment, which means that the SYSCALL environment is automatically initialized. A difference between the REXX support and shell scripts is that a REXX EXEC can be invoked from a C program, while a shell script can only be interpreted from the shell. A REXX EXEC can be called from a shell script.

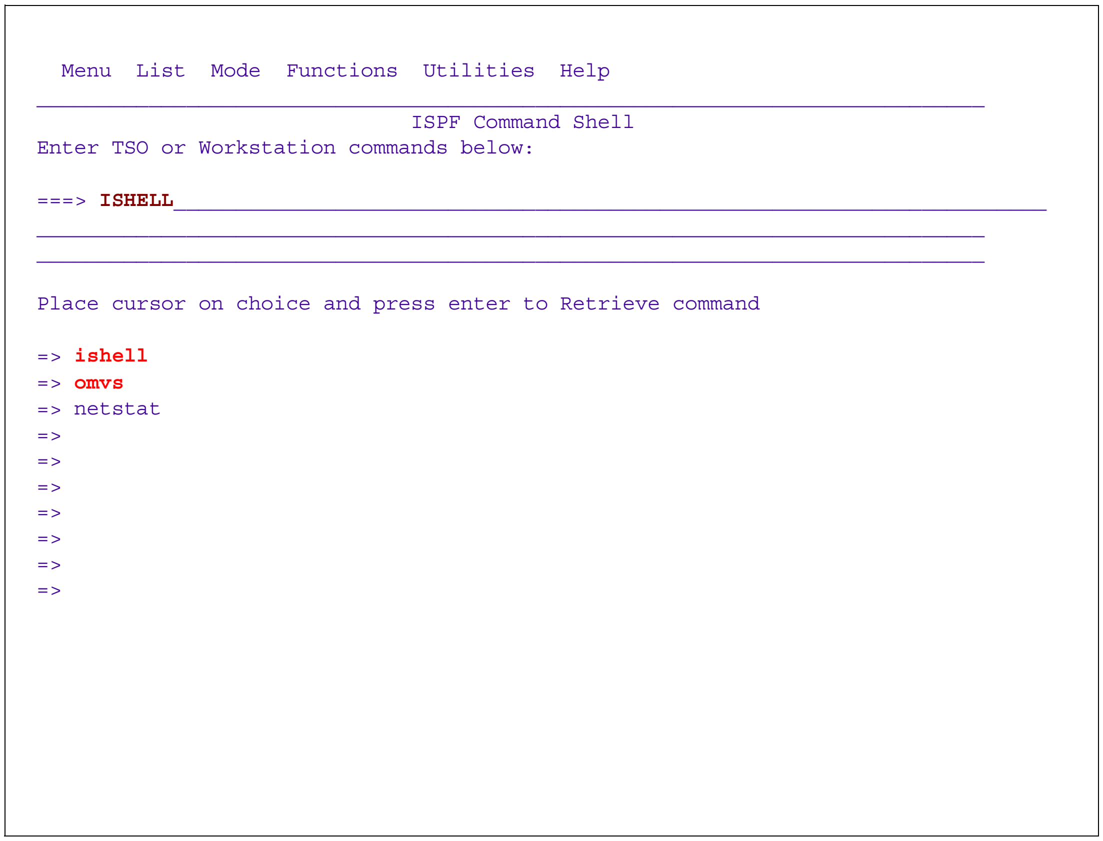
An interactive user who uses the OMVS command to access the shell can switch back and forth between the shell and TSO/E, the interactive interface to MVS.

Programmers whose primary interactive computing environment is a UNIX or AIX workstation find the z/OS shell programming environment familiar.

Programmers whose primary interactive computing environment is TSO/E and ISPF can do much of their work in that environment.

Interactive users of z/OS UNIX have a choice between using a UNIX-like interface (the shell), a TSO interface (TSO commands), or an ISPF interface (ISPF CUA dialog). With these choices, users can choose the interface which they are most familiar with and get a quicker start on z/OS UNIX.

2.17 ISPF Option 6



```
  Menu  List  Mode  Functions  Utilities  Help
_____________________________________________________________________________
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> ISHELL__________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

Place cursor on choice and press enter to Retrieve command

=> ishell
=> omvs
=> netstat
=>
=>
=>
=>
=>
=>
=>
```

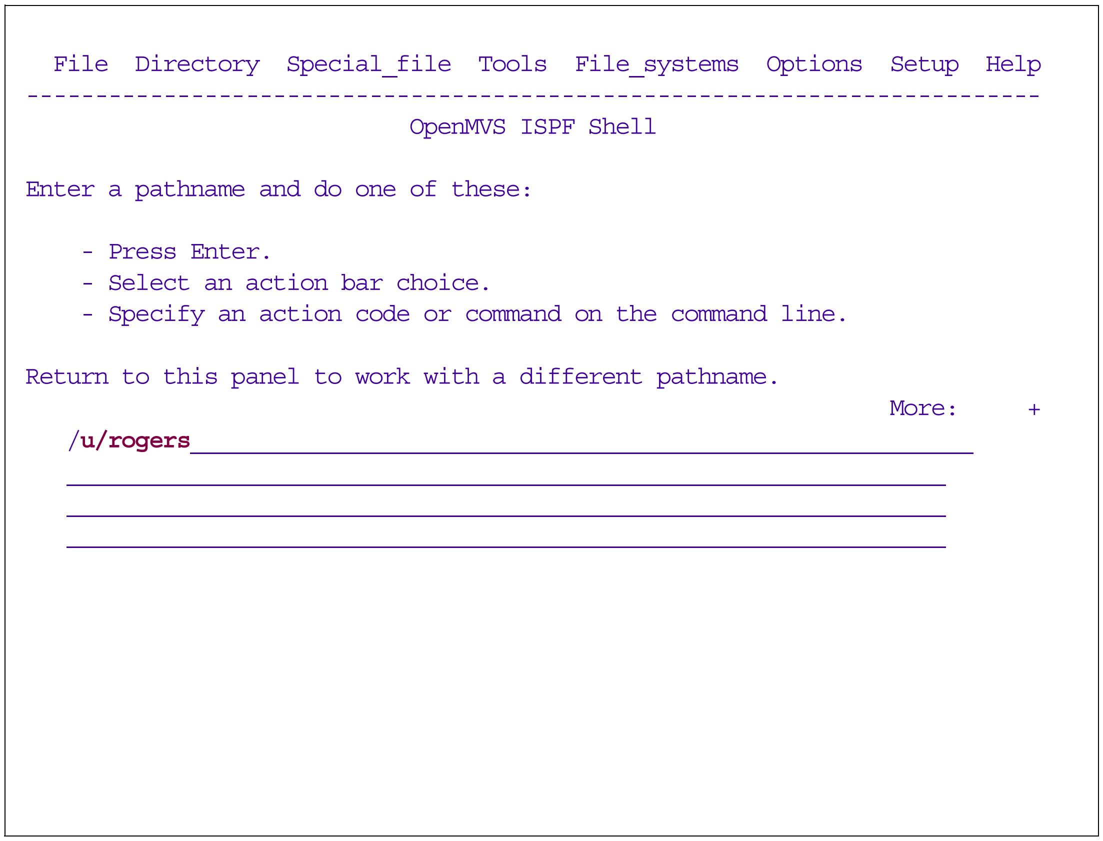
Figure 2-17 ISPF Option 6 panel

After a logon to TSO/E, enter Option 6 under ISPF to use the OMVS command and the ISHELL command.

If you are a user with an MVS background, you may prefer to use the ISPF shell panel interface instead of shell commands or TSO/E commands to work with the file system. The ISPF shell also provides the administrator with a panel interface for setting up users for z/OS UNIX access, for setting up the root file system, and for mounting and unmounting a file system.

You can also run shell commands, REXX programs, and C programs from the ISPF shell. The ISPF shell can direct stdout and stderr only to an HFS file, not to your terminal. If it has any contents, the file is displayed when the command or program completes.

2.18 ISHELL command (ish)



```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 ---------------------------------------------------------------------------
                            OpenMVS ISPF Shell

 Enter a pathname and do one of these:

     - Press Enter.
     - Select an action bar choice.
     - Specify an action code or command on the command line.

 Return to this panel to work with a different pathname.
                                                            More:     +
    /u/rogers__________________________________________________________
    _________________________________________________________________
    _________________________________________________________________
    _________________________________________________________________
```

Figure 2-18   Panel displayed after issuing the ish command

This is the ISHELL or ISPF Shell panel displayed as a result of the **ISHELL** or **ish** command being entered from ISPF Option 6.

To search a user's files and directories, type the following and press Enter:

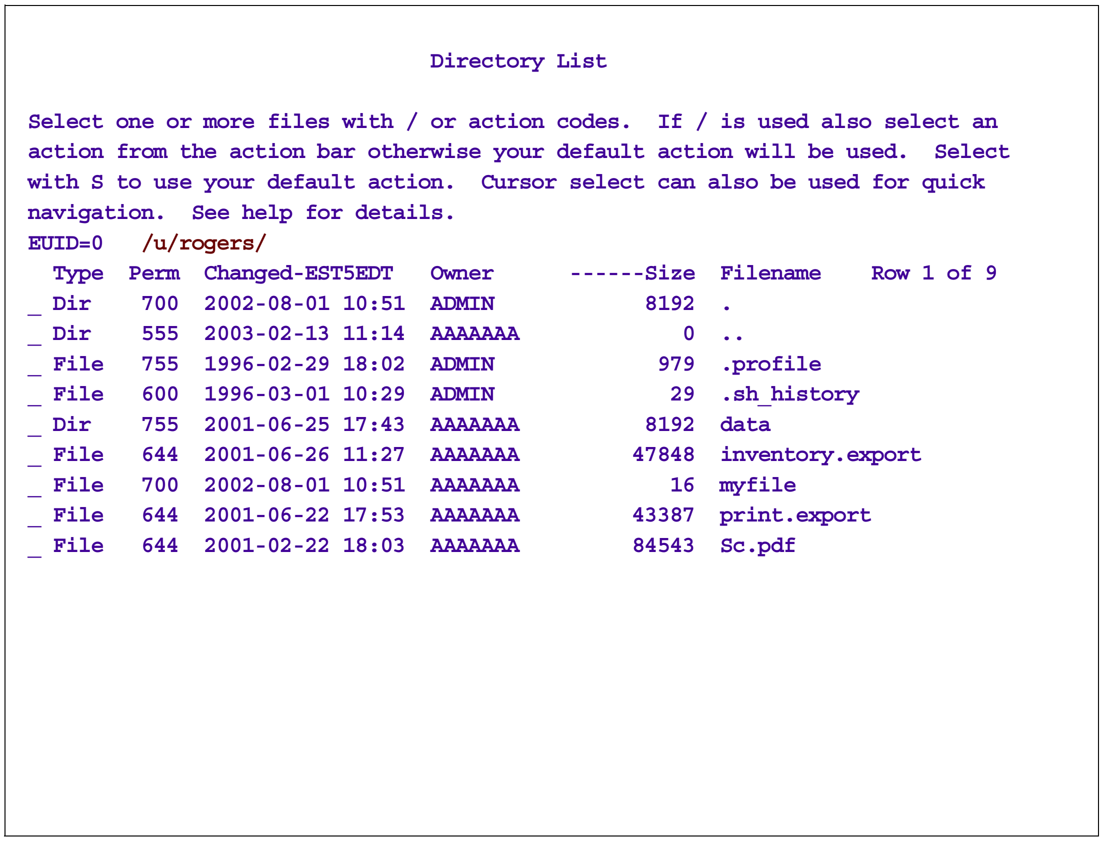
```
/u/userid
```

At the top of the panel is the action bar, with seven choices:

- File
- Directory
- Special file
- Tools
- File systems
- Options
- Setup
- Help

When you select one of these choices, a pulldown panel with a list of actions is displayed.

2.19 User's files and directories



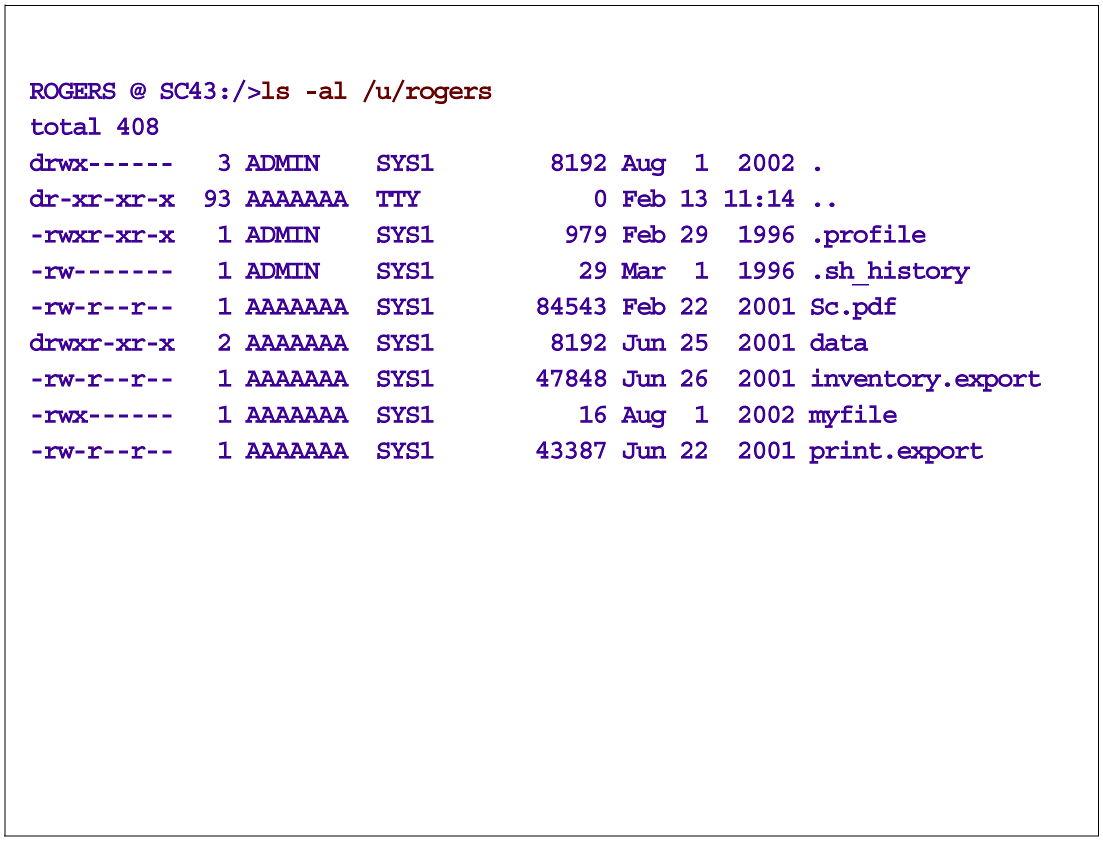
```
                                Directory List

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /u/rogers/
  Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 9
_ Dir    700  2002-08-01 10:51  ADMIN            8192  .
_ Dir    555  2003-02-13 11:14  AAAAAAA             0  ..
_ File   755  1996-02-29 18:02  ADMIN             979  .profile
_ File   600  1996-03-01 10:29  ADMIN              29  .sh_history
_ Dir    755  2001-06-25 17:43  AAAAAAA          8192  data
_ File   644  2001-06-26 11:27  AAAAAAA         47848  inventory.export
_ File   700  2002-08-01 10:51  AAAAAAA            16  myfile
_ File   644  2001-06-22 17:53  AAAAAAA         43387  print.export
_ File   644  2001-02-22 18:03  AAAAAAA         84543  Sc.pdf
```

Figure 2-19 Display of a user's files and directories

Shown in the figure are the files and directories of user **rogers**. The user can then use action codes to do the following:

b	Browse a file or directory
e	Edit a file or directory
d	Delete a file or directory
r	Rename a file or directory
a	Show the attributes of a file or directory
c	Copy a file or directory

2.20 OMVS command shell session



```
ROGERS @ SC43:/>ls -al /u/rogers
total 408
drwx------    3 ADMIN    SYS1        8192 Aug  1  2002 .
dr-xr-xr-x   93 AAAAAAA  TTY            0 Feb 13 11:14 ..
-rwxr-xr-x    1 ADMIN    SYS1         979 Feb 29  1996 .profile
-rw-------    1 ADMIN    SYS1          29 Mar  1  1996 .sh_history
-rw-r--r--    1 AAAAAAA  SYS1       84543 Feb 22  2001 Sc.pdf
drwxr-xr-x    2 AAAAAAA  SYS1        8192 Jun 25  2001 data
-rw-r--r--    1 AAAAAAA  SYS1       47848 Jun 26  2001 inventory.export
-rwx------    1 AAAAAAA  SYS1          16 Aug  1  2002 myfile
-rw-r--r--    1 AAAAAAA  SYS1       43387 Jun 22  2001 print.export
```

Figure 2-20 OMVS shell session display after issuing the OMVS command

Use the OMVS command to invoke the z/OS shell. Once you are working in a shell session, you can switch to subcommand mode, return temporarily to TSO/E command mode, or end the session by exiting the shell.

Shell commands often have options (also known as flags) that you can specify, and they usually take an argument, such as the name of a file or directory. The format for specifying the command begins with the command name, then the option or options, and finally the argument, if any.

In Figure 2-20, the following command is shown:

```
ls -al /u/rogers
```

where **ls** is the command name, **-al** are the options.

This figure shows the screen when the OMVS command is issued from ISPF Option 6. This command lists the files and directories of the user. If the pathname is a file, **ls** displays information on the file according to the requested options. If it is a directory, **ls** displays information on the files and subdirectories therein. You can get information on a directory itself using the **-d** option.

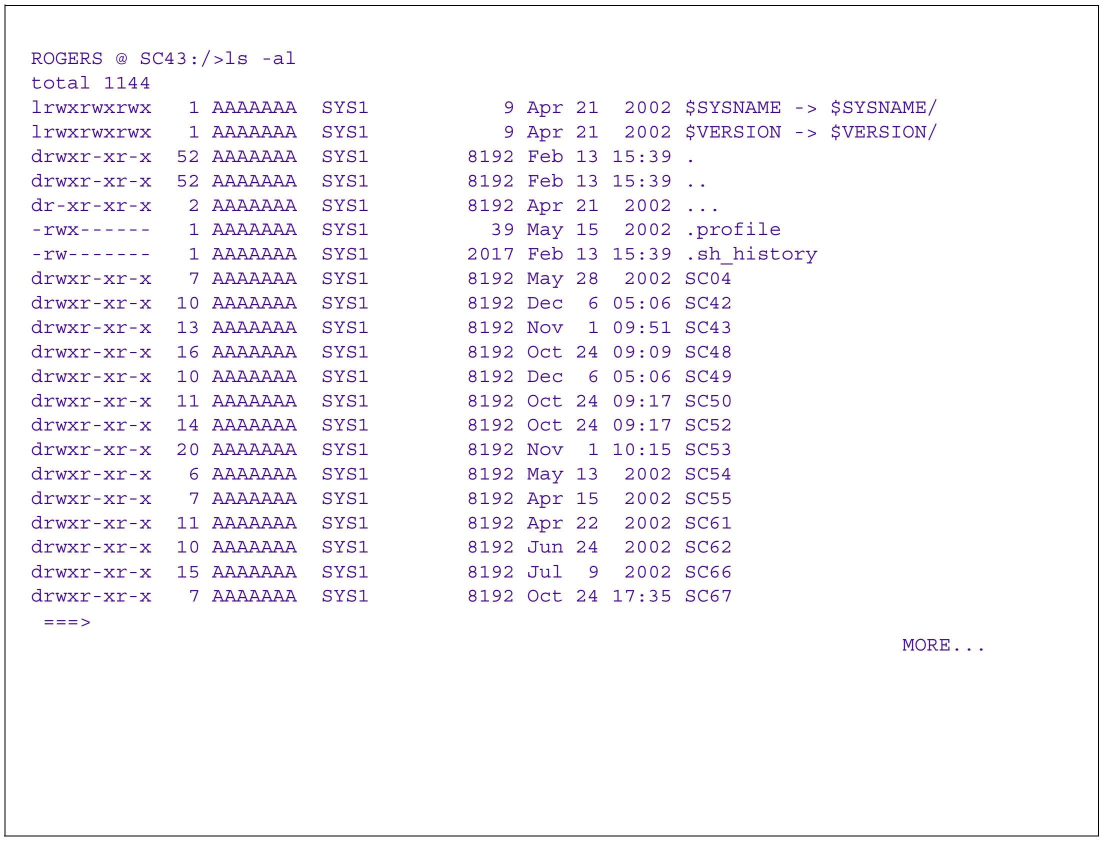
The shell is a command processor that you use to:

- Invoke shell commands or utilities that request services from the system.
- Write shell scripts using the shell programming language.

- Run shell scripts and C-language programs interactively (in the foreground), in the background, or in batch.

If you do not specify any options, **`ls`** displays only the filenames. When **`ls`** sends output to a pipe or a file, it writes one name per line; when it sends output to the terminal, it uses the **`-C`** (multicolumn) format.

2.21 ls -al command - list files in the root



```
ROGERS @ SC43:/>ls -al
total 1144
lrwxrwxrwx   1 AAAAAAA  SYS1           9 Apr 21  2002 $SYSNAME -> $SYSNAME/
lrwxrwxrwx   1 AAAAAAA  SYS1           9 Apr 21  2002 $VERSION -> $VERSION/
drwxr-xr-x  52 AAAAAAA  SYS1        8192 Feb 13 15:39 .
drwxr-xr-x  52 AAAAAAA  SYS1        8192 Feb 13 15:39 ..
dr-xr-xr-x   2 AAAAAAA  SYS1        8192 Apr 21  2002 ...
-rwx------   1 AAAAAAA  SYS1          39 May 15  2002 .profile
-rw-------   1 AAAAAAA  SYS1        2017 Feb 13 15:39 .sh_history
drwxr-xr-x   7 AAAAAAA  SYS1        8192 May 28  2002 SC04
drwxr-xr-x  10 AAAAAAA  SYS1        8192 Dec  6 05:06 SC42
drwxr-xr-x  13 AAAAAAA  SYS1        8192 Nov  1 09:51 SC43
drwxr-xr-x  16 AAAAAAA  SYS1        8192 Oct 24 09:09 SC48
drwxr-xr-x  10 AAAAAAA  SYS1        8192 Dec  6 05:06 SC49
drwxr-xr-x  11 AAAAAAA  SYS1        8192 Oct 24 09:17 SC50
drwxr-xr-x  14 AAAAAAA  SYS1        8192 Oct 24 09:17 SC52
drwxr-xr-x  20 AAAAAAA  SYS1        8192 Nov  1 10:15 SC53
drwxr-xr-x   6 AAAAAAA  SYS1        8192 May 13  2002 SC54
drwxr-xr-x   7 AAAAAAA  SYS1        8192 Apr 15  2002 SC55
drwxr-xr-x  11 AAAAAAA  SYS1        8192 Apr 22  2002 SC61
drwxr-xr-x  10 AAAAAAA  SYS1        8192 Jun 24  2002 SC62
drwxr-xr-x  15 AAAAAAA  SYS1        8192 Jul  9  2002 SC66
drwxr-xr-x   7 AAAAAAA  SYS1        8192 Oct 24 17:35 SC67
 ===>
                                                                        MORE...
```

Figure 2-21 A UNIX command that lists the files in the root

This figure shows the result of the **`ls -al`** command. It displays the files and directories and shows the file access allowed for the user, the user's group, and other users.

2.22 Direct login to shell

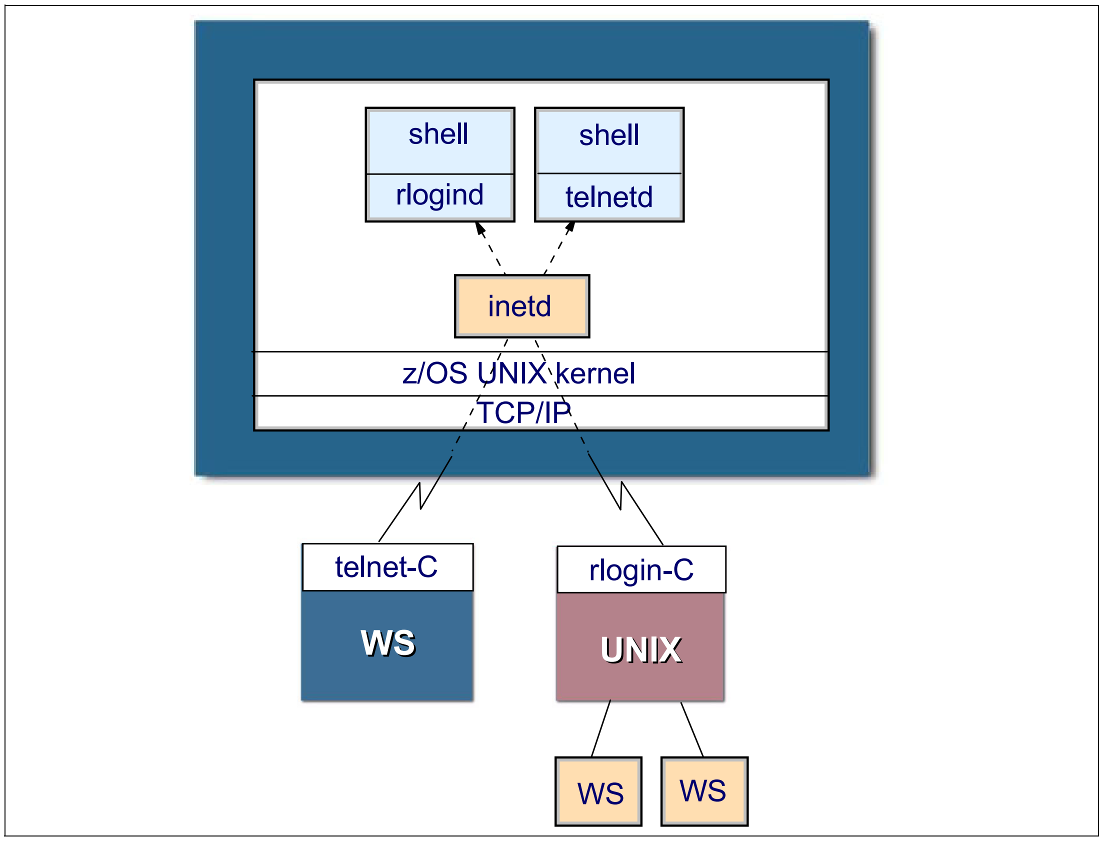


Figure 2-22 Diagram of a login to the shell from a terminal workstation

A z/OS UNIX user can log in directly to the z/OS UNIX shell using one of the following solutions:

rlogin When the inetd daemon is set up and active, you can rlogin to the shell from a workstation that has rlogin client support and is connected via TCP/IP or Communications Server to the MVS system. To log in, use the rlogin (remote log in) command syntax supported at your site.

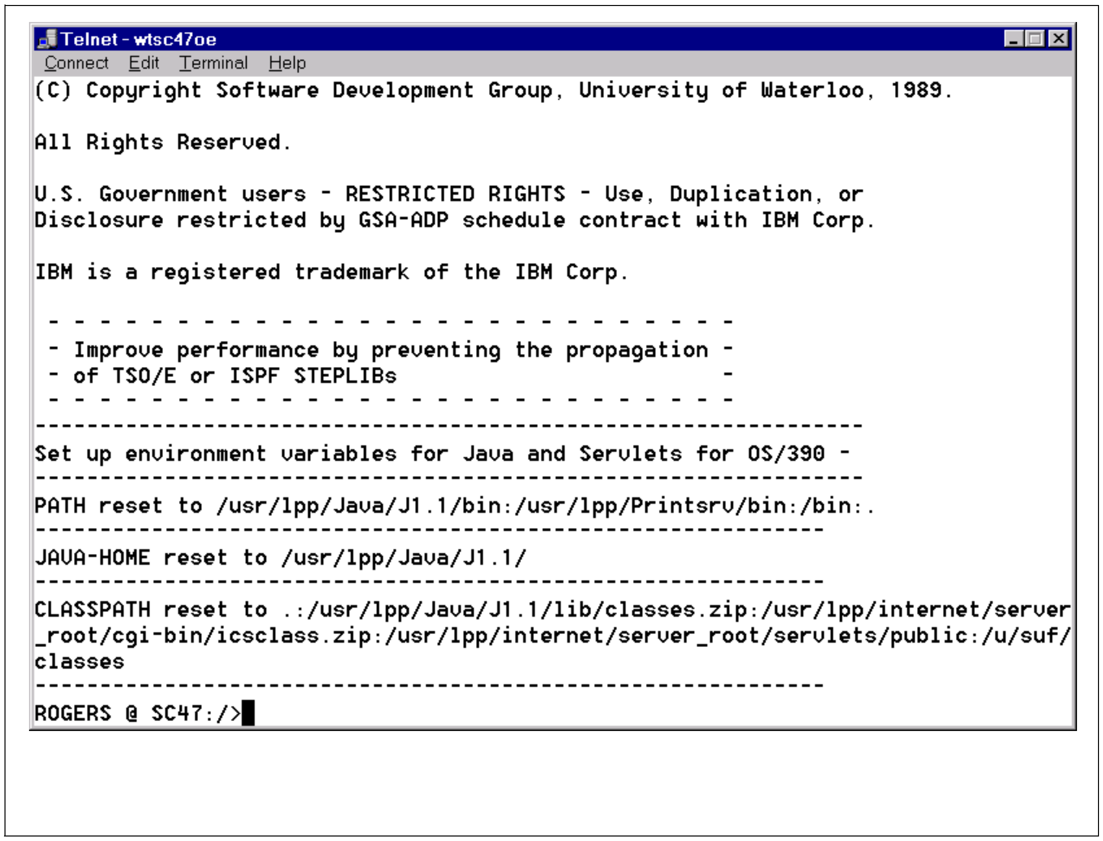
telnet The telnet support comes with the TCP/IP z/OS UNIX feature. It also uses the inetd daemon, which must be active and set up to recognize and receive the incoming telnet requests.

There are some differences between the asynchronous terminal support (direct shell login) and the 3270-terminal support (OMVS command):

- You cannot switch to TSO/E. However, you can use the TSO shell command to run a TSO/E command from your shell session.
- You cannot use the ISPF editor (this includes the oedit and TSO/E OEDIT commands, which invoke ISPF edit).

The TCP/IP chapters have more details about this. It is mentioned here to give you a picture of the methods available to invoke the shell. Later chapters also have more information about differences between the 3270 and the direct login methods.

2.23 Telnet access to z/OS UNIX



Telnet - wtsc47oe

Connect Edit Terminal Help

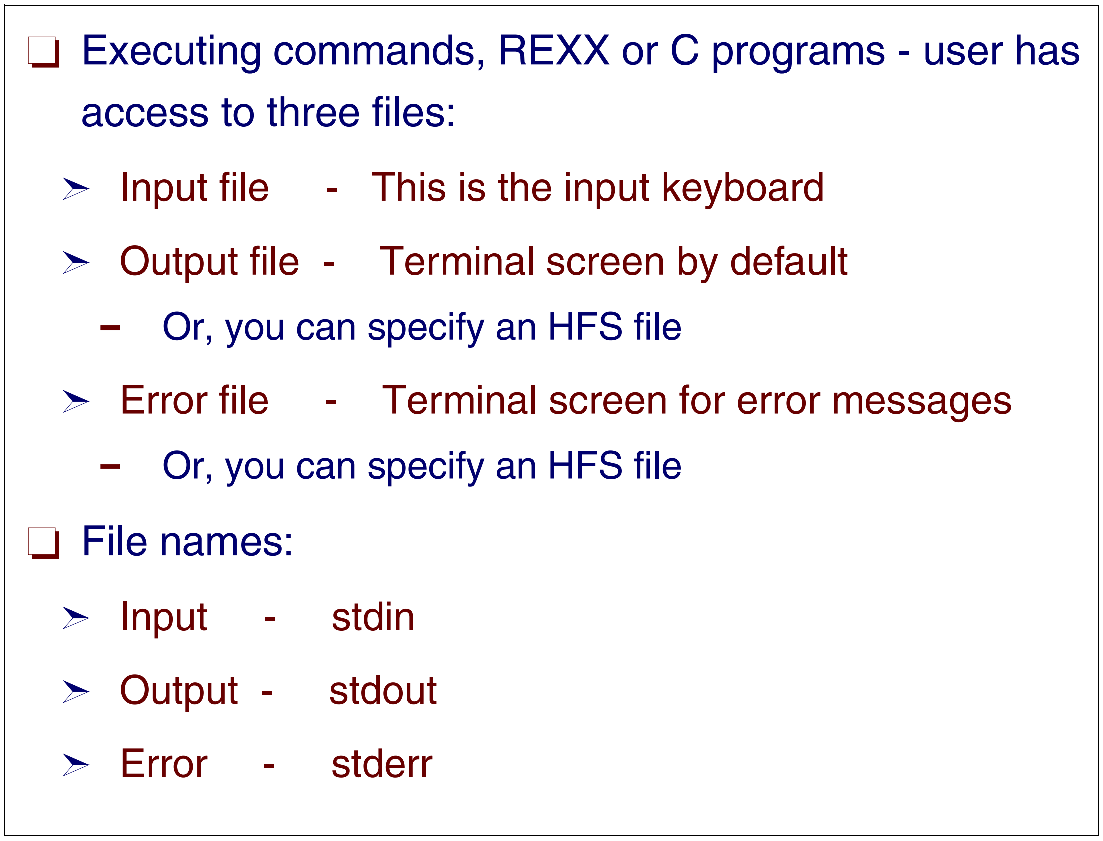
```
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

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                          -
 - - - - - - - - - - - - - - - - - - - - - - - - - - -
----------------------------------------------------------------
Set up environment variables for Java and Servlets for OS/390 -
----------------------------------------------------------------
PATH reset to /usr/lpp/Java/J1.1/bin:/usr/lpp/Printsrv/bin:/bin:.
-------------------------------------------------------------
JAVA-HOME reset to /usr/lpp/Java/J1.1/
-------------------------------------------------------------
CLASSPATH reset to .:/usr/lpp/Java/J1.1/lib/classes.zip:/usr/lpp/internet/server
_root/cgi-bin/icsclass.zip:/usr/lpp/internet/server_root/servlets/public:/u/suf/
classes
-------------------------------------------------------------
ROGERS @ SC47:/>
```

Figure 2-23 Telnet login to the shell screen

This figure shows the z/OS shell after being entered from a telnet login.

From this session, you cannot switch to TSO/E or use the ISPF editor.

2.24 Input, output, errors with UNIX shell



- Executing commands, REXX or C programs - user has access to three files:
  - Input file - This is the input keyboard
  - Output file - Terminal screen by default
    - Or, you can specify an HFS file
  - Error file - Terminal screen for error messages
    - Or, you can specify an HFS file
- File names:
  - Input - stdin
  - Output - stdout
  - Error - stderr

Figure 2-24 Files that a user has access to in a shell session

z/OS C/C++ programs require that stdin, stdout, and stderr be defined as either a file or a terminal. Many C functions use stdin, stdout, and stderr. However, it depends on how the application is coded whether or not it writes to stderr and stdout.

When a shell command begins running, it has access to three files:

- It reads from its standard input file. By default, standard input is the keyboard.
- It writes to its standard output file.
  - If you invoke a shell command from the shell, a C program, or a REXX program invoked from TSO READY, standard output is directed to your terminal screen by default.
  - If you invoke a shell command, REXX program, or C program from the ISPF shell, standard output cannot be directed to your terminal screen. You can specify an HFS file or use the default, a temporary file.
- It writes error messages to its standard error file.
  - If you invoke a shell command from the shell or from a C program or from a REXX program invoked from TSO READY, standard error is directed to your terminal screen by default.
  - If you invoke a shell command, REXX program, or C program from the ISPF shell, standard error cannot be directed to your terminal screen. You can specify an HFS file or use the default, a temporary file.

If the standard output or standard error file contains any data when the command completes, the file is displayed for you to browse.

In the shell, the names for these files are:

- stdin for the standard input file
- stdout for the standard output file
- stderr for the standard error file

2.25 BPXBATCH utility

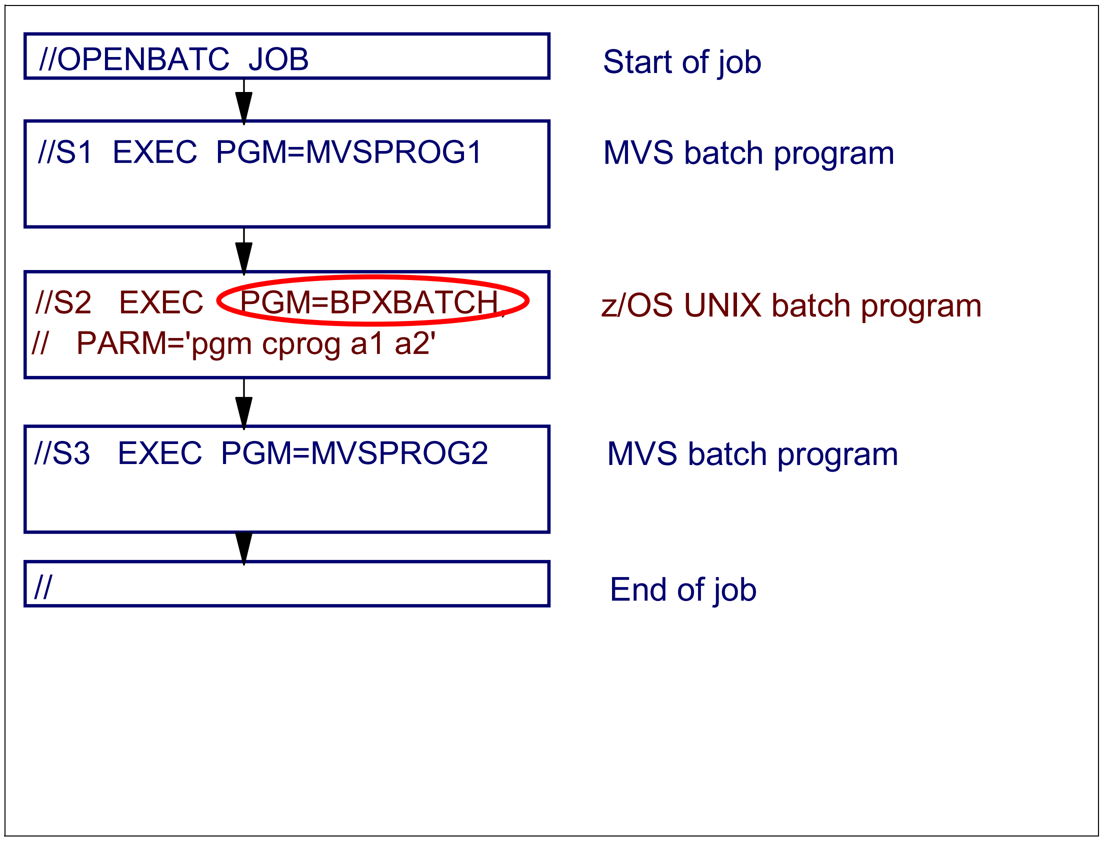


Figure 2-25 JCL used when using the BPXBATCH utility

BPXBATCH is an MVS utility that you can use to run shell commands or shell scripts and to run executable files through the MVS batch environment. You can invoke BPXBATCH:

- In JCL
- From the TSO/E READY prompt
- From TSO CLISTs and REXX execs
- From a program

BPXBATCH has logic in it to detect when it is run from JCL. If the BPXBATCH program is running as the only program on the job step task level, it sets up the stdin, stdout, and stderr and execs the requested program. If BPXBATCH is not running as the only program at the job step task level, the requested program will run as the second step of a JES batch address space from JCL in batch.

If run from any other environment, the requested program will run in a WLM initiator in the OMVS subsys category.

2.26 BPXBATCH job

```
//OMVSPGM   JOB   USER=userid
//OMVSEXEC  EXEC   PGM=BPXBATCH,PARM='pgm cprog a1 a2'
//STDOUT      DD   PATH='/dir1/dir2/std.output',
//                 PATHOPTS=(OWRONLY,OCREATE),
//                 PATHMODE=(SIRWXV),
//                 PATHDISP=KEEP
//STDERR      DD   PATH='/dir1/dir2/std.error',
//                 PATHOPTS=(OWRONLY,OCREATE),
//                 PATHMODE=(SIRWXV),
//                 PATHDISP=KEEP
/*
```

Figure 2-26   Sample JCL for a BPXBATCH job

BPXBATCH makes it easy for you to run, from your TSO/E session, shell scripts or z/OS C executable files that reside in hierarchical file system (HFS) files.

With BPXBATCH, you can allocate the MVS standard files stdin, stdout, and stderr as HFS files for passing input. If you do allocate these files, they must be HFS files. You can also allocate MVS data sets or HFS text files containing environment variables (stdenv). If you do not allocate them, stdin, stdout, stderr, and stdenv default to /dev/null. Allocate the standard files using the data definition PATH keyword options, or standard data definition options for MVS data sets.

The BPXBATCH default for stderr is the same file defined for stdout. For example, if you define stdout to be /tmp/output1 and do not define stderr, then both printf() and perror() output is directed to /tmp/output1.

For BPXBATCH, you can define stdin, stdout, and stderr using one of the following:

- The TSO/E ALLOCATE command, using the ddnames STDIN, STDOUT, and STDERR.
- A JCL DD statement with the PATH operand, using the ddnames STDIN, STDOUT, and STDERR.
- Redirection.

2.27 The dbx debugger

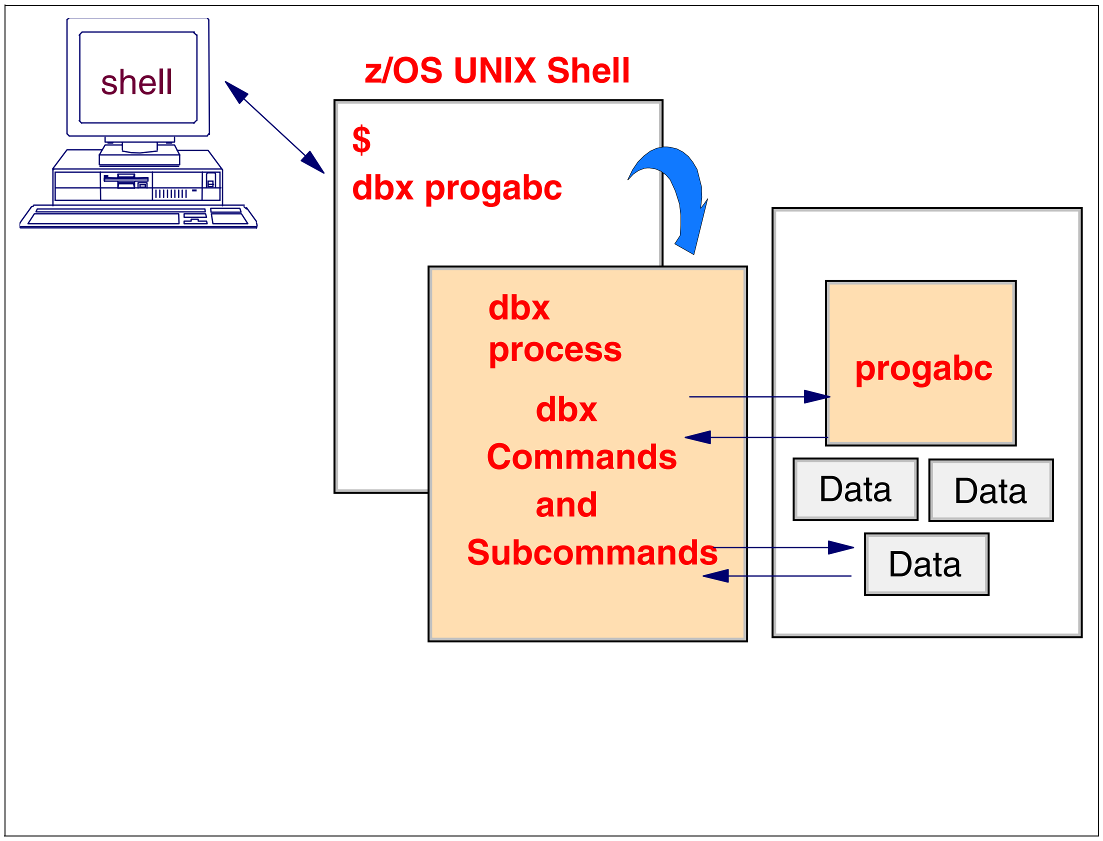


Figure 2-27 z/OS UNIX dbx debugger

The z/OS UNIX dbx debugger is an interactive tool for debugging C language programs that use z/OS UNIX. It is based upon the dbx debugger, which is regarded as an industry standard on UNIX systems.

The dbx debugger provides the options to debug at source level or assembler level. Source-level debugging allows you to debug C language programs. Assembler-level debugging allows you to debug executable programs at machine code level.

The dbx debugger is a utility that is invoked from the z/OS UNIX shell. It cannot be invoked directly from TSO/E. In the shell, dbx is the debugging facility for z/OS C/C++ programs. With dbx, you can debug multithreaded applications at the C-source level or at the machine level. Support for multithreaded applications gives you the ability to:

- Debug or display information about the following objects related to multithreaded applications: threads, mutexes, and condition variables.
- Control program execution by holding and releasing individual threads.

When using the interactive dbx debugger you can:

- Run a program one line or one instruction at a time.
- Set breakpoints at selected statements and machine instructions with conditions for activation.
- Access variables symbolically and display them in the correct format.
- Display or modify the contents of registers, variables, and storage.

- Examine the source text using simple search functions.
- Debug processes that contain fork() and exec() functions.
- Interrupt and examine a program that is already in progress.
- Trace processing of a one-task program by line, instruction, routine, or variable.
- Call programs or diagnostic routines directly from the debug program.
- Invoke shell commands from debug session.
- Examine loaded address maps for a process.

3

UNIX System Services Pre-Installation

This chapter describes the necessary pre-installation requirements to get UNIX System Services up and running.

It explains in detail the following topics:

- The difference between ServerPac and CBPDO
- The RACF definitions needed before initializing UNIX System Services
- Allocation of the root file
- The TSO/E debugging necessary when using the z/OS UNIX ISHELL

3.1 Installing z/OS using ServerPac

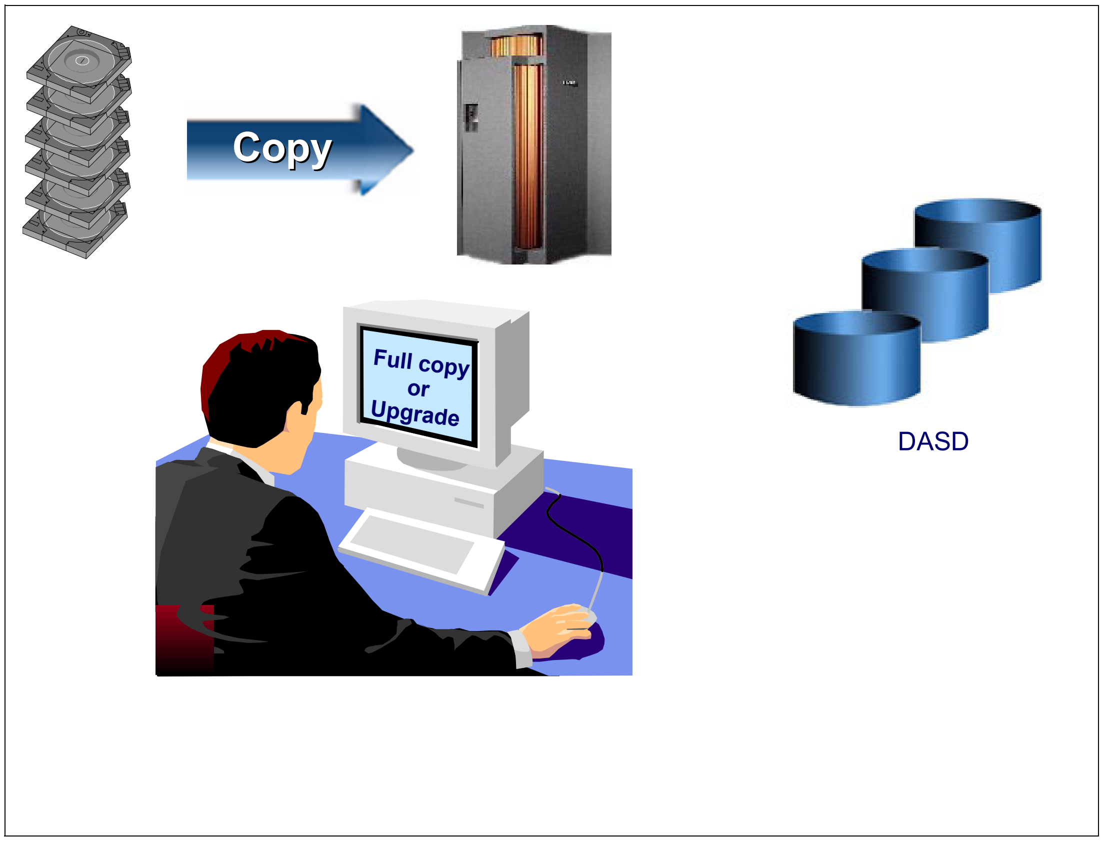


Figure 3-1 Installing z/OS using the ServerPac

ServerPac is a software delivery package consisting of products and services for which IBM has performed the SMP/E installation steps and some of the post-SMP/E installation steps. To install the package on your system and complete the installation of the software it includes, you use the CustomPac Installation Dialog, the same dialog that is used for all the CustomPac offerings, including SystemPac® and ProductPac®.

Two types of ServerPac installation are available:

- A full system replacement
- A software upgrade

Note: A full system replacement will provide system software related data sets like Dlib and Target, as well as SMP/E CSI data sets and sample libraries. A software upgrade can only be done if those data sets are available.

A full system replacement installs all the data sets needed to IPL, log on to the target system, and run a z/OS image in order to complete other installation and customization tasks.

A software upgrade installs only system software and related data sets, while preserving your existing operational data sets. An upgrade uses your existing catalog structure. It is only possible for z/OS, not for subsystems.

3.2 Installing z/OS using CBPDO

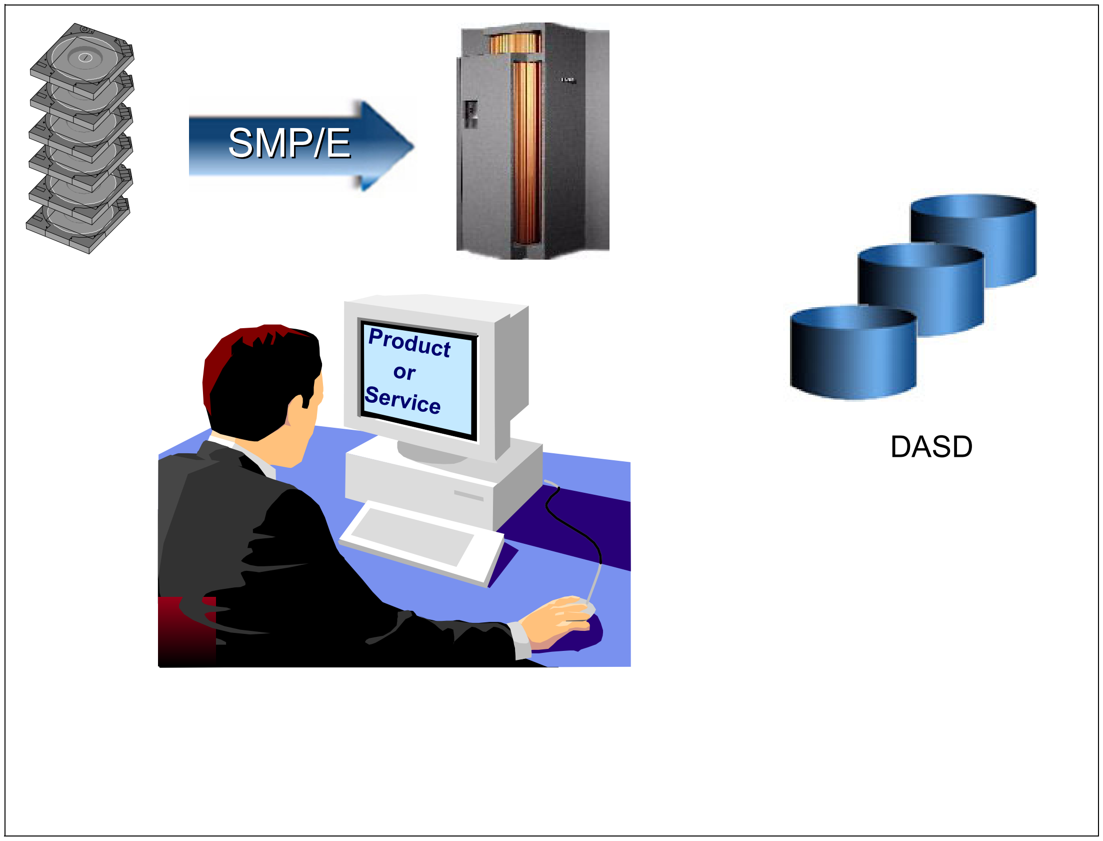


Figure 3-2 Installing z/OS using CBPDO

Custom-Built Product Delivery Option (CBPDO) is a software delivery package consisting of uninstalled products and unintegrated service. You must use SMP/E to install the individual z/OS elements and features, and their service, before you can IPL.

The CBPDO installation process is split into separate stages, called waves. These waves group related activities together so that at the completion of each wave, the wave components can be activated (like performing an IPL).

These waves are:

- Wave® 0 installs prerequisite FMIDs onto the driving system, such as HLASM and SMP/E.
- Wave 1 installs FMIDs that do not install into HFS.
- Wave 2 installs FMIDs that do install into HFS.
- Wave 3 installs FMIDs for JES2 or JES3.

3.3 UNIX System Services installation

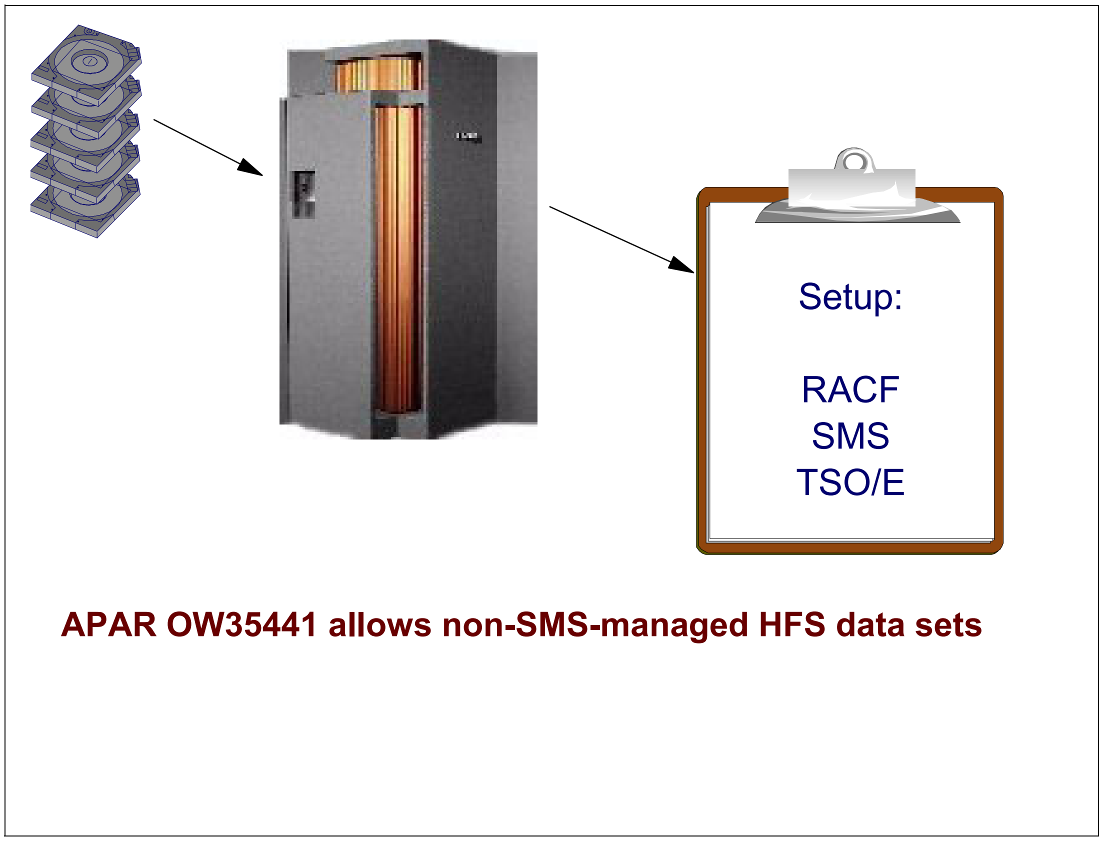


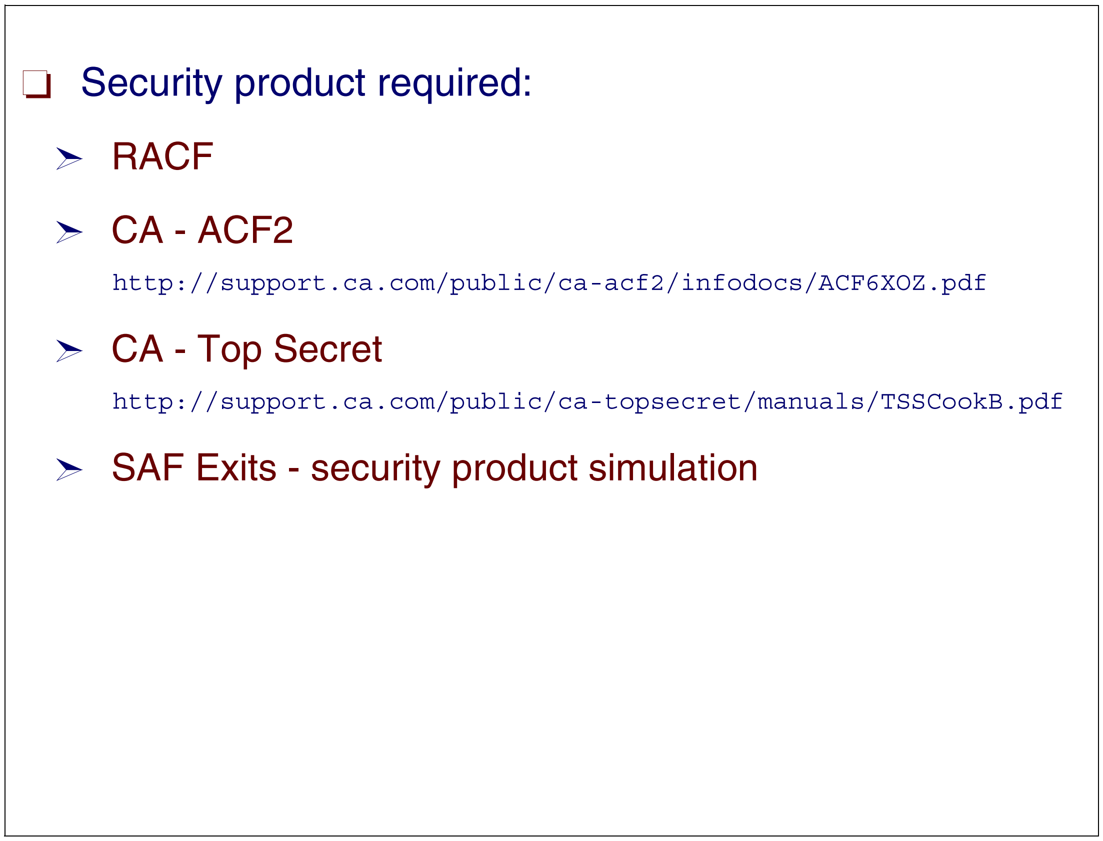
Figure 3-3 Components needed for z/OS UNIX installation

The ServerPac provides you with full system replacement and software upgrade options. Online panels contain the jobs and present the information you need to proceed through the ServerPac installation.

Before you can install and set up UNIX System Services, the following must be set up in your environment:

- RACF
- SMS
- TSO/E

3.4 z/OS UNIX security



- Security product required:
  - RACF
  - CA - ACF2

    http://support.ca.com/public/ca-acf2/infodocs/ACF6XOZ.pdf
  - CA - Top Secret

    http://support.ca.com/public/ca-topsecret/manuals/TSSCookB.pdf
  - SAF Exits - security product simulation

Figure 3-4 z/OS UNIX security products

z/OS UNIX requires that some kind of security product be implemented. If you do not have a security product installed, then SAF exits need to be written to simulate a security product and give suitable responses to SAF calls.

RACF stores z/OS UNIX data in OMVS segments, but Solution Developer products such as CA-ACF2 and CA-Top Secret have also implemented solutions to seamlessly support z/OS UNIX. Implementation cookbooks are available for those Solution Developer products.

For CA-ACF2, see:

`http://support.ca.com/public/ca-acf2/infodocs/ACF6XOZ.pdf`

For CA-Top Secret, see:

`http://support.ca.com/public/ca-topsecret/manuals/TSSCookB.pdf`

The cookbooks available for both ACF2 and Top Secret z/OS UNIX should work just as well under those products as RACF.

3.5 RACF definitions

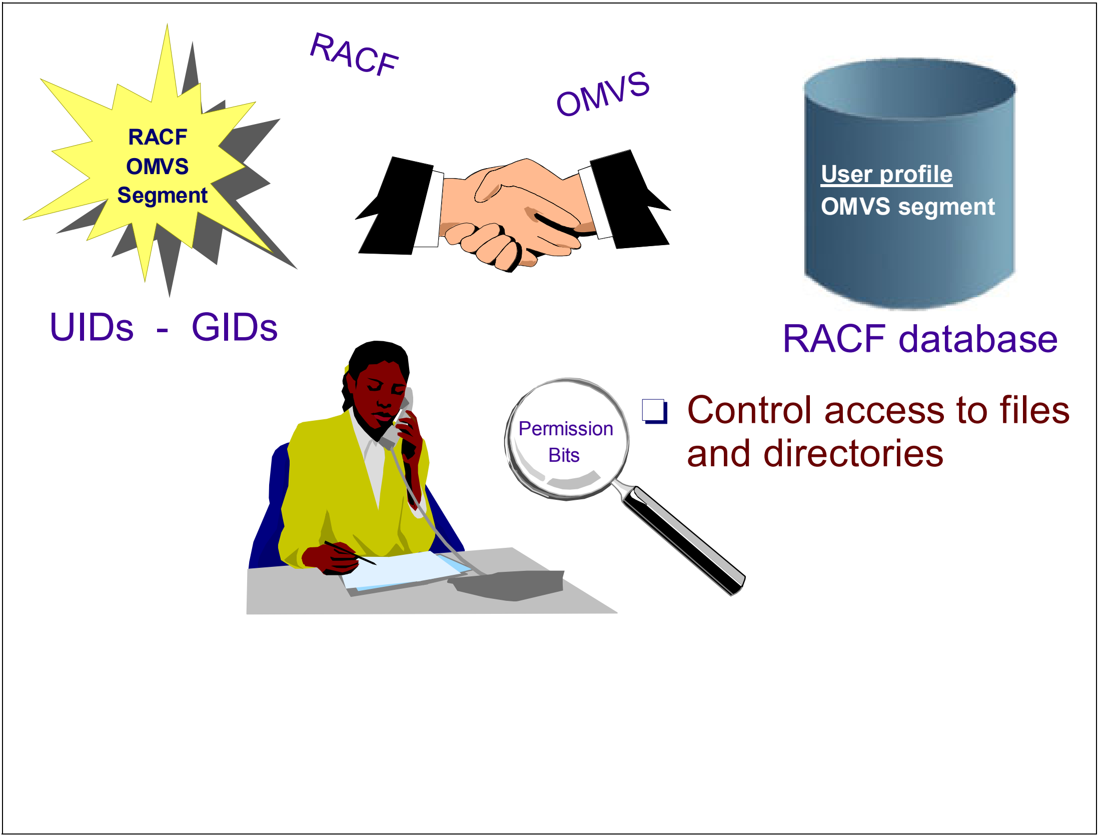


Figure 3-5 RACF definitions for z/OS UNIX

z/OS UNIX provides security mechanisms that work with the security offered by the z/OS system. A security product is required.

Before you can install and debug UNIX System Services you need to have access to UNIX System Services data sets and members which are named in directories and files.

RACF and other security-related products allow access to your MVS UNIX environment. This access can be allowed without being a TSO/E user.

If your user ID should have access to z/OS UNIX, your security administrator has to specify the home directory, the shell program, and a UID in the OMVS segment. In addition, the administrator has to provide a group ID (GID) for any RACF group the user is connected to. If this is done you should be able to access the UNIX shell or work with the ISPF-driven ISHELL. The decision whether you will be able to access directories or files will be made by UNIX security.

In addition, your RACF administrator has to provide a user ID that is assigned to the OMVS and BPXOINIT address spaces (Started Procedure). This user ID needs only to have an OMVS segment and should be connected to a RACF group with a GID.

3.6 RACF OMVS segments

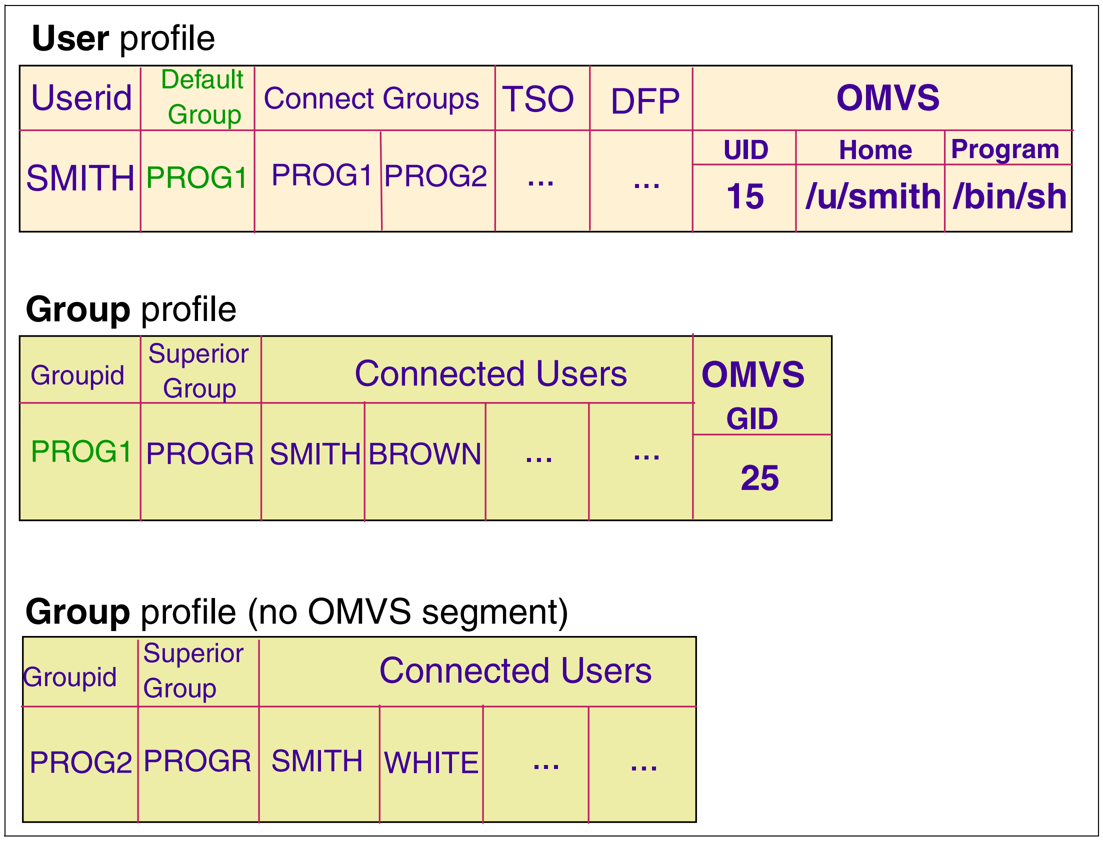


Figure 3-6 z/OS UNIX RACF OMVS segments

All users and programs that need access to z/OS UNIX System Services must have a RACF user profile defined, with an OMVS segment which has at least a UID specified. A user without a UID cannot access z/OS UNIX.

Note: It is possible for multiple users to have the same UID number specified. However, this is not recommended.

RACF user profile

The RACF user profile has a segment called OMVS for z/OS UNIX support. A user ID must have an OMVS segment defined in order to use UNIX System Services, for example, access the ISHELL or the shell. This segment has three fields, as follows:

UID	A number from 0 to 2147483647 that identifies a z/OS UNIX user. A z/OS UNIX user must have a UID defined.
Home	The name of a directory in the file system. This directory is called the home directory. This field is optional.
Program	The name of a program. This is the program that will be started for the user when the user begins a z/OS UNIX session. Usually this is the program name for the z/OS UNIX shell. This field is optional.

RACF group profile

The RACF group also has a segment called OMVS to define z/OS UNIX groups. It contains only one field, as follows:

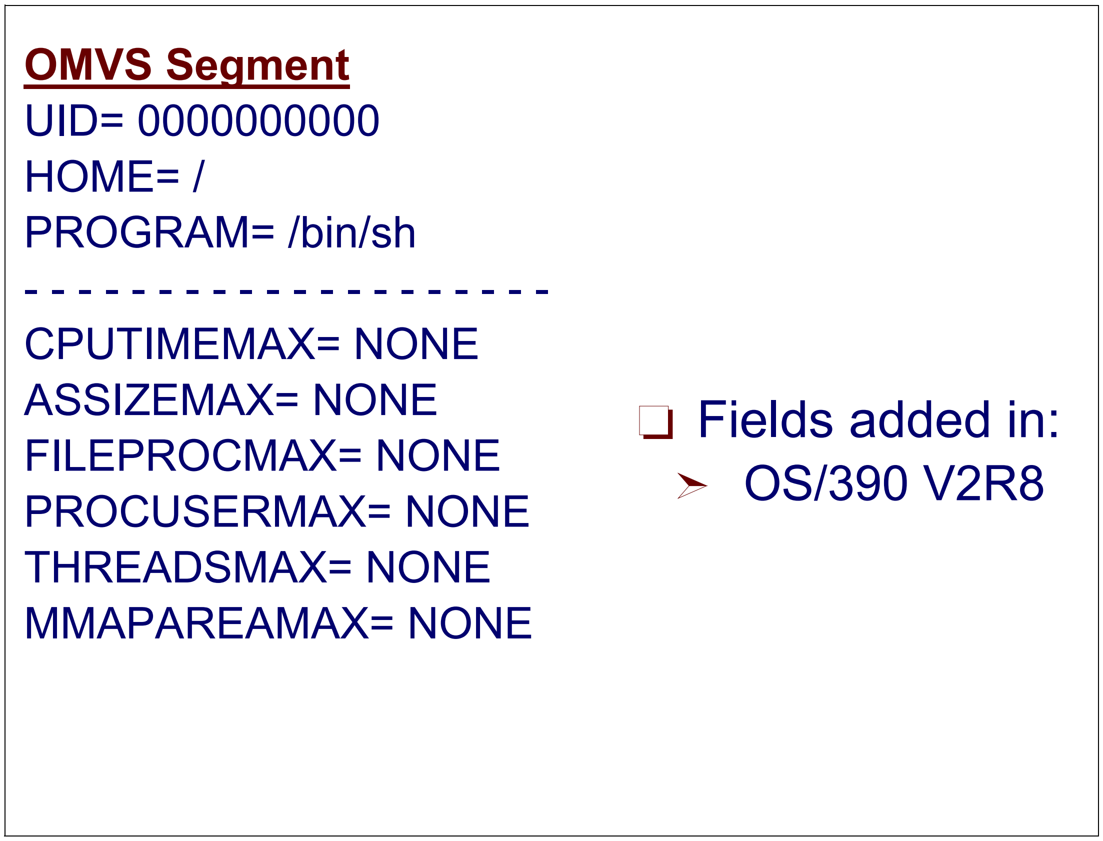
GID A number from 0 to 2147483647 which identifies a z/OS UNIX group.

The home directory is the current directory when a user invokes z/OS UNIX. During z/OS UNIX processing this can be changed temporarily by using the `cd` (change directory) shell command. The command will not change the value in the RACF profile. The directory specified as home directory in the RACF profile must exist (be pre-allocated) before a user can invoke z/OS UNIX. If a home directory is not specified in RACF, the root (/) directory will be used as default.

The example in Figure 3-6 shows a user profile for TSO/E user ID SMITH which is connected to two groups, PROG1 and PROG2. SMITH is defined as a z/OS UNIX user because he has a UID specified. His home directory is **/u/smith** and he will get into the shell when he issues the OMVS command because the name of the shell, **/bin/sh** is specified as program name.

The program name in the OMVS segment specifies the name of the first program to start when z/OS UNIX is invoked. Usually this is the name of the z/OS UNIX shell.

3.7 OMVS segment fields



OMVS Segment

UID= 0000000000
HOME= /
PROGRAM= /bin/sh
- -
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE

- Fields added in:
  - OS/390 V2R8

Figure 3-7 Fields in the OMVS segment

ASSIZEMAX MAXASSIZE is the maximum region size (in bytes) for an address space. 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 ASSIZEMAX limit on a per-user basis as follows:

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

CPUTIMEMAX MAXCPUTIME is the time limit (in seconds) for processes that were created by rlogind 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 user basis as follows:

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

FILEPROCMAX Use MAXFILEPROC to determine the number of character-special files, /dev/fdxx, that a single process can have open concurrently. You can also limit the amount of system resources available to a single user process. 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 FILEPROCMAX limit on a per user basis as follows:

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

MMAPAREAMAX For MAXMMAPAREA, 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 MMAPAREAMAX limit on a per user basis as follows:

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

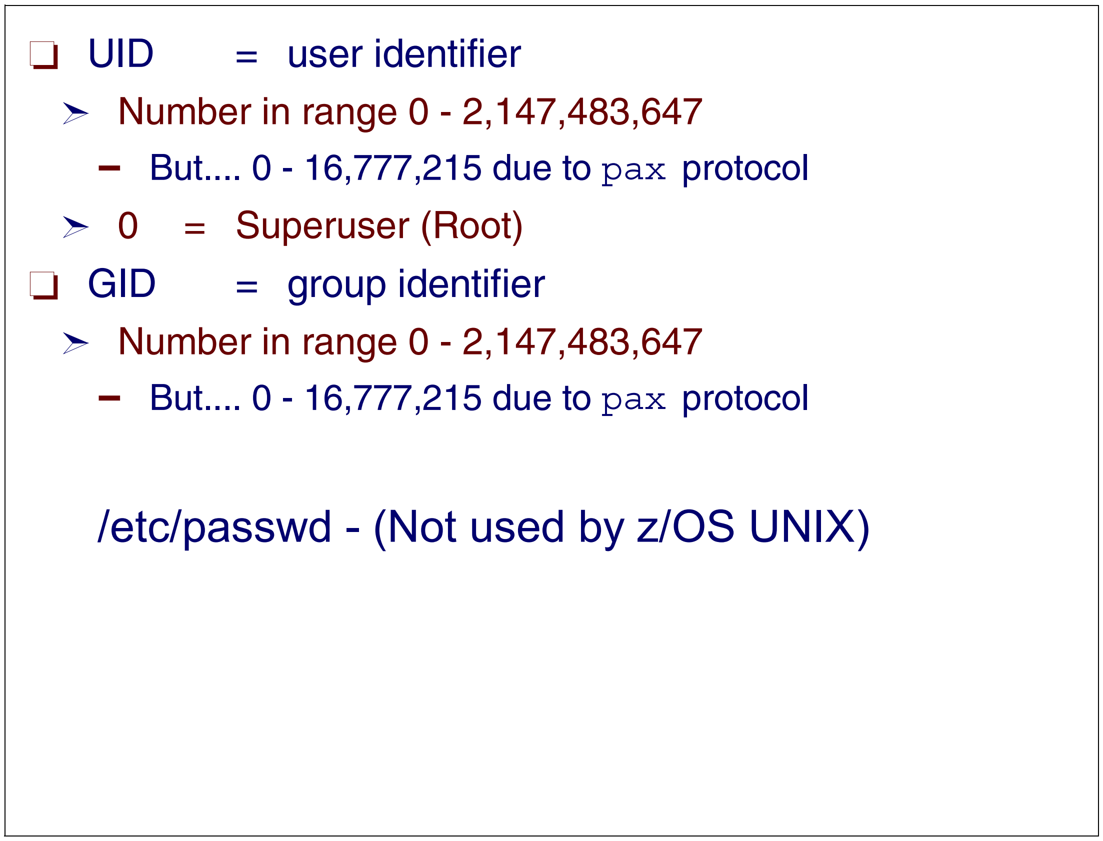
PROCUSERMAX You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the PROCUSERMAX limit on a per-user basis as follows:

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

THREADSMAX 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 16M line by specifying C run-time options. 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))
```

3.8 UNIX security



- UID = user identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol
  - 0 = Superuser (Root)
- GID = group identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol

/etc/passwd - (Not used by z/OS UNIX)

Figure 3-8 Security on UNIX platforms

UNIX systems incorporate a concept of users and groups similar to that of RACF. A user UNIX identifier (or UID) is a number between 0 and some large number that varies between brands of UNIX. User numbers do not have to be unique and it is possible (though not recommended) for several users to share the same UID, and even be logged on at the same time. UNIX sees these users as being the same entities and they receive the same levels of authorization. A user with UID=0 is what's called a superuser (ROOT on some brands of UNIX). The superuser has unlimited authority within a UNIX environment. For obvious reasons, UID=0 needs to be strictly controlled.

Users are all related to a group. Groups allow authority to be controlled in a more economical way, in that giving access to a group is a lot easier than giving access to a couple of hundred users. A group identifier (or GID) is a number between 0 and some large number that varies between brands of UNIX. Any number of users can share the same GID.

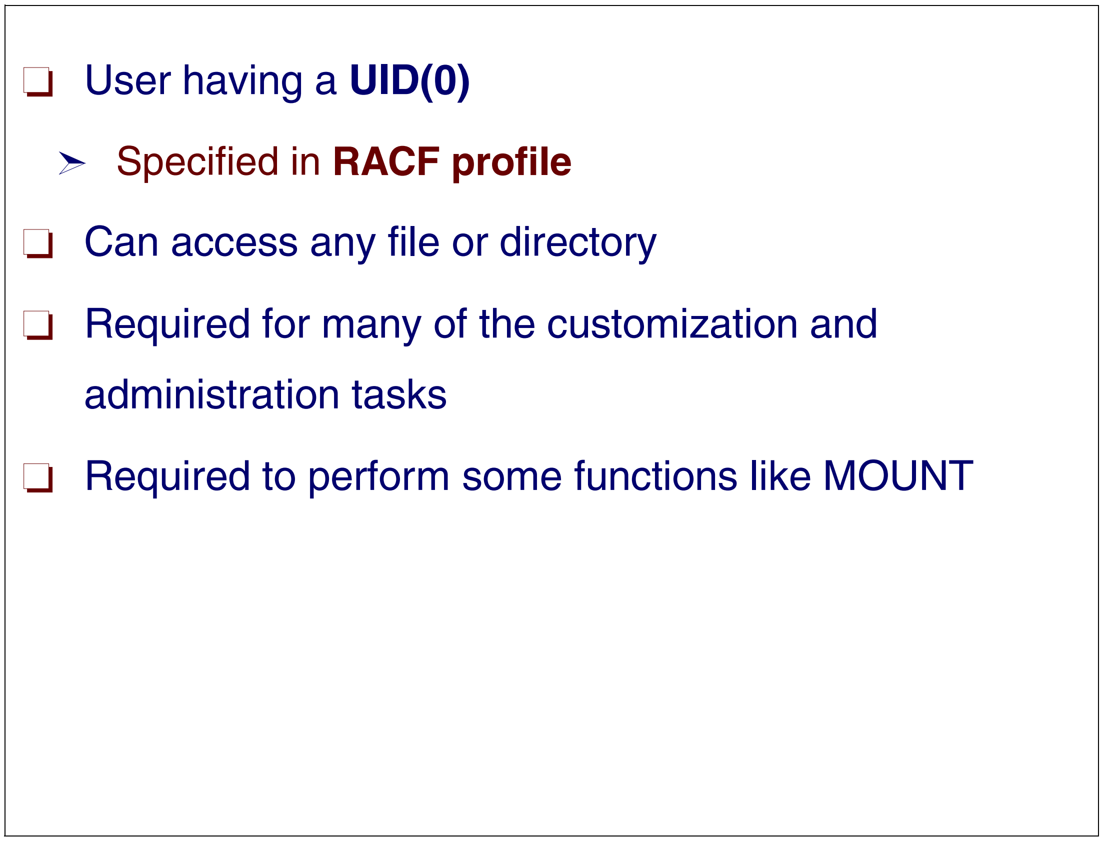
`pax` and **`tar`** are UNIX utilities that perform functions like PKZIP and DF/DSS. The z/OS implementation of UNIX allows UID and GID numbers up to 2,147,483,647. The pax protocol supports UIDs and GIDs up to 8 octal digits. The largest value supported is 77777777 octal or 16M decimal. To allow the use of pax and tar, keep the UIDs and GIDs below this value.

It is recommended that the maximum number used should be no more than 77,777,777.

UNIX systems typically store their user/group information in a file called:

```
/etc/passwd
```

3.9 Superuser



- User having a **UID(0)**
  - Specified in **RACF profile**
- Can access any file or directory
- Required for many of the customization and administration tasks
- Required to perform some functions like MOUNT

Figure 3-9 z/OS UNIX and superusers

z/OS UNIX has no predefined numbering scheme for UIDs and GIDs, with the exception of UID=0. UID=0 is a superuser (also known as ROOT on other UNIX platforms) and has the ability to bypass all forms of security within the UNIX environment.

A superuser is a user with a UID of 0 and this UID is in the RACF profile of the user. A superuser can be a started procedure with a trusted or privileged attribute defined in the RACF STARTED Class or added to the RACF Started Procedures Table. The procedure must have a UID. However, when a started procedure is trusted or privileged, RACF does not base authority checking on the UID value; the UID can have any value.

A superuser can do the following:

- Pass all security checks, so that the superuser can access any file in the file system.
- Perform the mount function.
- Change identity from one UID to another.
- Manage processes.
- Have an unlimited number of processes 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.
- Change identity from one UID to another.
- Use setrlimit to increase any of the system limits for a process.

3.10 RACF commands and user IDs

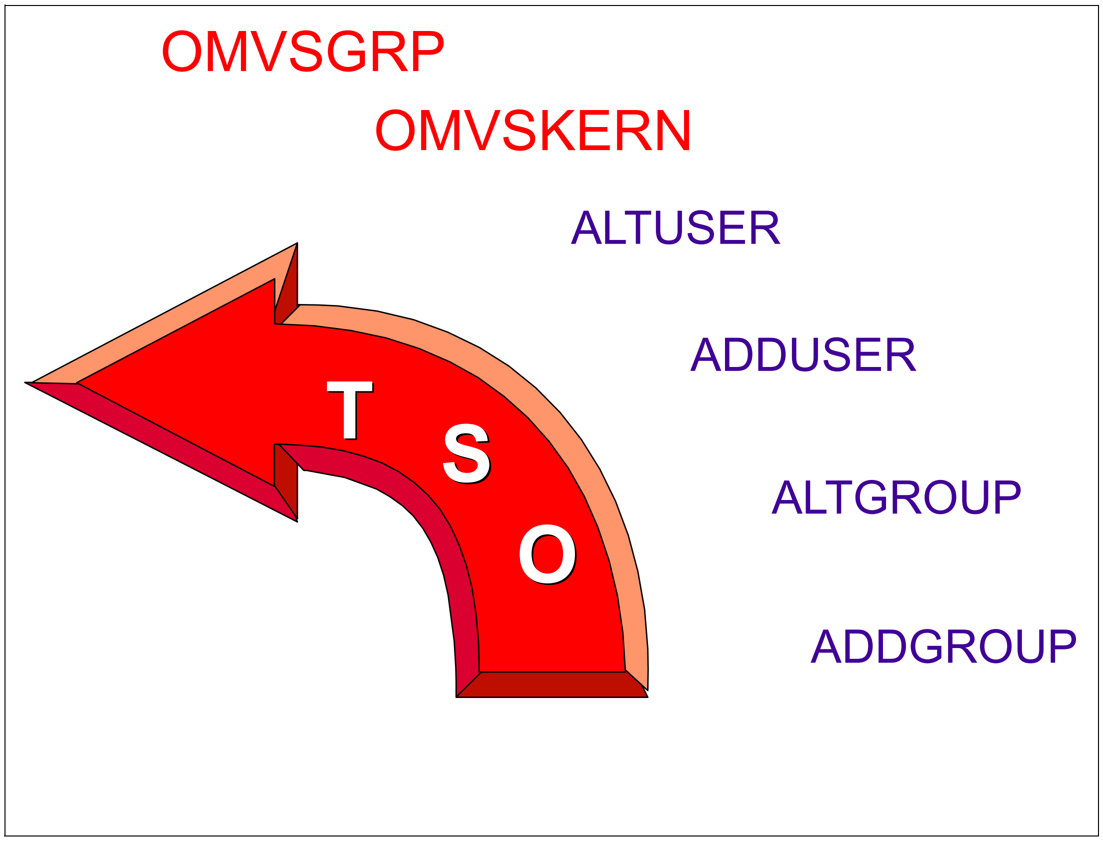


Figure 3-10 The RACF commands used to define users and groups

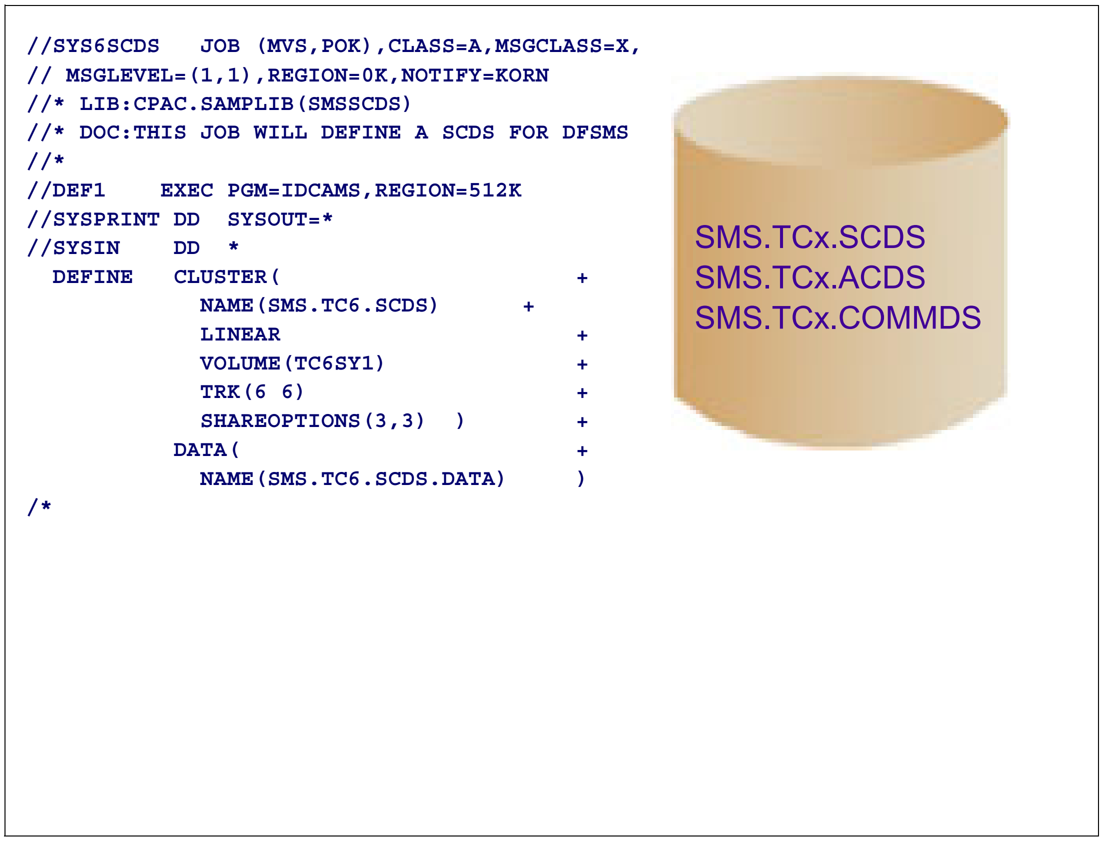
For our UNIX System Service installation we should provide the following setup in RACF:

- Provide a GID for our default RACF group.
- Define the OMVS RACF segment.
- Define the OMVSKERN user ID and group.
- Define the OMVS and BPXOINIT cataloged procedures.

The following RACF commands can be used to fulfill the UNIX RACF prerequisites:

```
ALTGROUP SYS1 OMVS(GID(0))
ALTUSER KORN OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
ADDGROUP OMVSGRP OMVS(GID(1))
ADDUSER OMVSKERN DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
PASSWORD(ONE6PACK).
RDEFINE STARTED OMVS.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(YES))
RDEFINE STARTED BPXOINIT.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
```

3.11 DFSMS definitions: Define SCDS



```
//SYS6SCDS   JOB (MVS,POK),CLASS=A,MSGCLASS=X,
// MSGLEVEL=(1,1),REGION=0K,NOTIFY=KORN
//* LIB:CPAC.SAMPLIB(SMSSCDS)
//* DOC:THIS JOB WILL DEFINE A SCDS FOR DFSMS
//*
//DEF1     EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  DEFINE   CLUSTER(                        +
             NAME(SMS.TC6.SCDS)        +
             LINEAR                        +
             VOLUME(TC6SY1)                +
             TRK(6 6)                      +
             SHAREOPTIONS(3,3)  )          +
           DATA(                           +
             NAME(SMS.TC6.SCDS.DATA)       )
/*
```

Figure 3-11 DFSMS definitions to define the SCDS

The Storage Management Subsystem (SMS) provides a range of data and space management functions. SMS improves storage space use, controls external storage centrally, and lets you manage storage growth. It makes it easier to convert to new device types, and takes advantage of what available hardware can do. With SMS, you can move toward system-managed storage.

Before you can activate an SMS configuration, you need to allocate control data sets used by SMS, and define their contents. Control data sets are virtual storage access method (VSAM) linear data sets that contain base configuration information, SMS class, aggregate group, optical library, tape library, optical drive, and storage group definitions and ACS routines.

Before the SMS address space is activated, you can allocate control data sets with access method services or TSO/E commands. You define and alter the contents of control data sets using ISMF. SMS uses three types of control data sets:

- A source control data set (SCDS)
- An active control data set (ACDS)
- A communications data set (COMMDS)

A SCDS contains an SMS configuration, which defines a storage management policy. You can define any number of SMS configurations, each of which has its own SCDS. Then, you select one SMS configuration to be the installation storage management policy and make an active working copy of it in an ACDS.

The COMMDS serves as the primary means of SMS communication among systems in the SMS complex. The active systems in an SMS complex access the COMMDS for current SMS complex information.

The COMMDS contains the name of the ACDS containing the currently active storage management policy, the current utilization statistics for each system-managed volume, and other system information. You can define any number of COMMDSs, but only one can be active in an SMS complex.

When you activate an SCDS, its contents are copied to an ACDS. The current ACDS contains a copy of the most recently activated configuration. All systems in an SMS complex use this configuration to manage storage. You can define any number of SCDSs, but only one can be put in the ACDS. You can define more than one IGDSMSmm member, each specifying a different ACDS, but you can use only one ACDS at a time.

3.12 DFSMS definitions

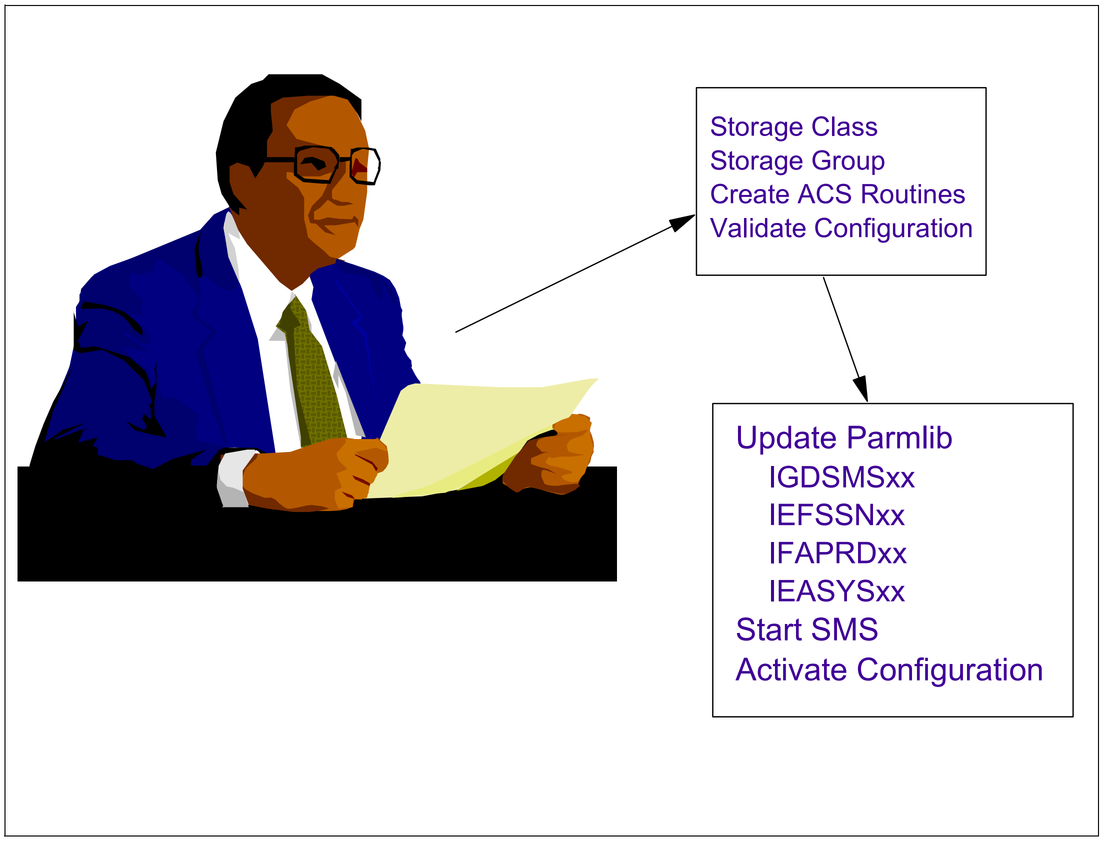


Figure 3-12 DFSMS definitions in SYS1.PARMLIB

After allocating an SCDS, you can define a base configuration. The first time you select ISMF, you get the ISMF Primary Option Menu for end users. To get the ISMF Primary Option Menu for storage administrators, select option 0, ISMF Profile. Within the ISMF Profile Option Menu, select option 0, User Mode, and press Enter. You get the User Mode Entry panel, where you indicate that you want the storage administrator Primary Option Menu for all future ISMF sessions. To do this, select option 2 on the User Mode Entry panel. After changing the user mode, you must exit ISMF and then return to it to view the Primary Option Menu for Storage Administrators.

Now you can go on defining SMS routines.

The IGDSMSmm, IEASYSyy, and IEFSSNxx members of SYS1.PARMLIB direct the initialization and activation of SMS. IGDSMSmm provides initialization parameters to SMS. The SMS=mm parameter of IEASYSyy indicates the name of the SYS1.PARMLIB member IGDSMSmm that is used for initialization. For example, if SMS=01 in IEASYSyy, then the IGDSMS01 member of SYS1.PARMLIB is used during initialization. The SMS entry in IEFSSNxx identifies SMS to MVS. The IFAPRDxx parmlib member is used to enable/disable z/OS optional elements.

After validation the SMS configuration must be activated, but this can only be done if the SMS subsystem is started.

3.13 DFSMS setup for z/OS UNIX

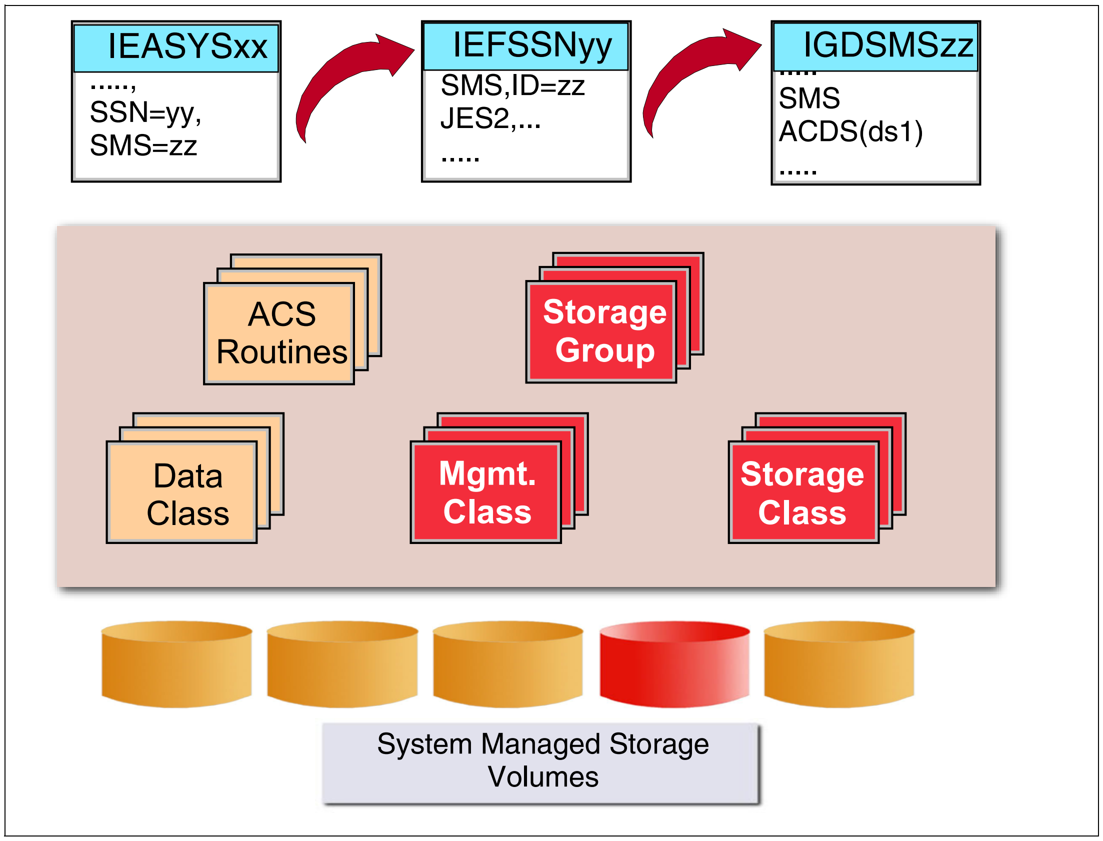


Figure 3-13 DFSMS system managed storage volumes

The hierarchical file system (HFS) data sets, which contain the UNIX file system files, are managed by DFSMS.

z/OS UNIX requires the following minimum configuration to be defined in the Storage Management Subsystem:

- One DASD volume to be SMS-managed
- One Management class
- One Storage class
- One Storage group

Note: APAR OW35441 now gives you the ability to allocate PDSE and HFS data sets on unmanaged (non-SMS) volumes, if running DFSMS 1.4 or DFSMS 1.5.

The following SYS1.PARMLIB members must be updated for SMS:

- IEFSSNyy defines the SMS as a subsystem and point to which IGDSMSzz member to use. Which IEFSSNxx member to use is specified by the SSN keyword in the IEASYSxx member that is used during IPL.
- IGDSMSzz contains the parameters that initialize the Storage Management Subsystem, for example the data set name of the Active Control Data Set (ACDS) which contains the SMS configuration.

3.14 DFSMS definitions

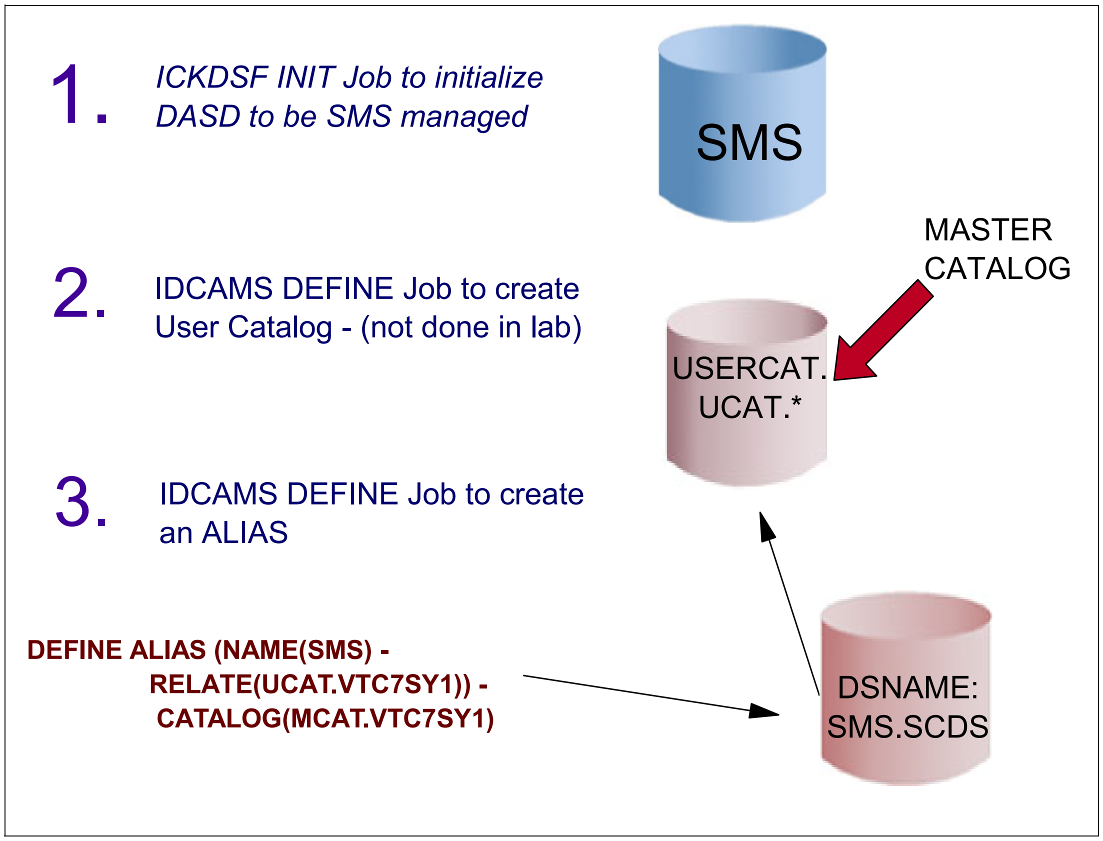

1. ICKDSF INIT Job to initialize DASD to be SMS managed
2. IDCAMS DEFINE Job to create User Catalog - (not done in lab)
3. IDCAMS DEFINE Job to create an ALIAS
DEFINE ALIAS (NAME(SMS) -
RELATE(UCAT.VTC7SY1)) -
CATALOG(MCAT.VTC7SY1)
SMS
MASTER CATALOG
USERCAT. UCAT.*
DSNAME: SMS.SCDS

Figure 3-14 Preparing the DASD for SMS storage groups

The first step toward activating SMS on the target system is the preparation of DASD for use as an SMS storage group. This storage group will be configured to contain HFS data sets (HFS data sets must be SMS-managed).

Note: APAR OW35441 now gives you the ability to allocate PDSE and HFS data sets on unmanaged (non-SMS) volumes, if running DFSMS 1.4 or DFSMS 1.5.

The Device Support Facility product ICKDSF can be used to initialize DASD to be SMS-managed. The INIT command of ICKDSF is used to define a volume serial (VOLSER) number and allocate an indexed VTOC. Specifying the STORAGEGROUP keyword on the INIT statement tells ICKDSF to make the DASD SMS-managed. Once initialized, the DASD can only be used to hold SMS-managed data sets.

The SMS control data sets (SCDS, ACDS and COMMDS) can be cataloged in the master catalog, but it is more flexible to have them created in a user catalog. If a suitable user catalog does not already exist, then IDCAMS can be used to create one.

Once a suitable user catalog is available, then an ALIAS needs to be defined to control what data sets get cataloged in the user catalog.

3.15 TSO/E support

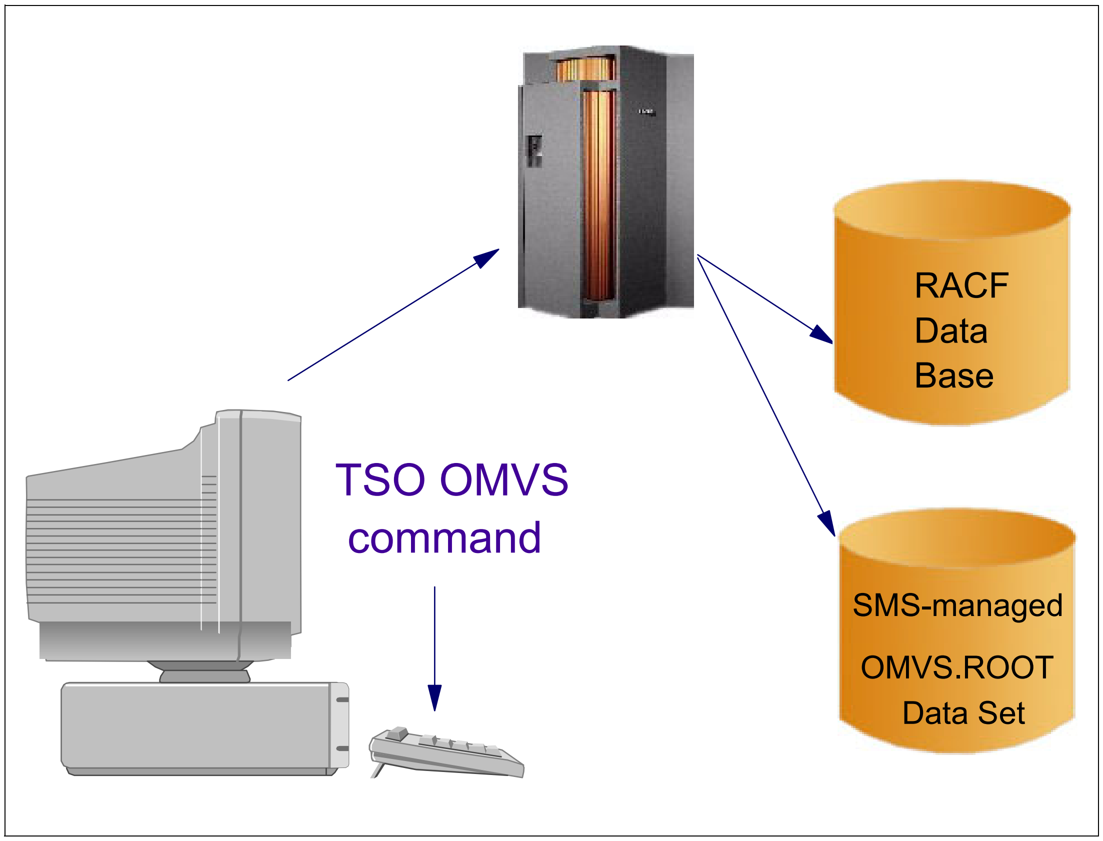


Figure 3-15 TSO/E component used with z/OS UNIX

One benefit to users of UNIX System Services is the UNIX System Services shell. The UNIX System Services shell is a command processor that you use to:

- Invoke shell commands or utilities that request services from the system (similar to TSO/E).
- Write shell scripts using the shell programming language (similar to REXX).
- Run shell scripts and C-language programs interactively in the foreground, in the background, or in batch (again, similar to REXX).

Once z/OS UNIX is installed, the shell can be invoked through the TSO/E command **OMVS**. Like the OBROWSE, OEDIT, and ISHELL commands, the OMVS command can also be added to an ISPF selection panel, or entered as a TSO/E command.

The z/OS UNIX ISPF Shell or ISHELL is an ISPF panel interface that you can use instead of TSO/E commands or shell commands to perform tasks against z/OS UNIX user IDs and HFSs.

If you are more comfortable using the ISPF editor and ISPF pull-down menus, the ISHELL is the tool for you.

3.16 User access to TSO/E commands

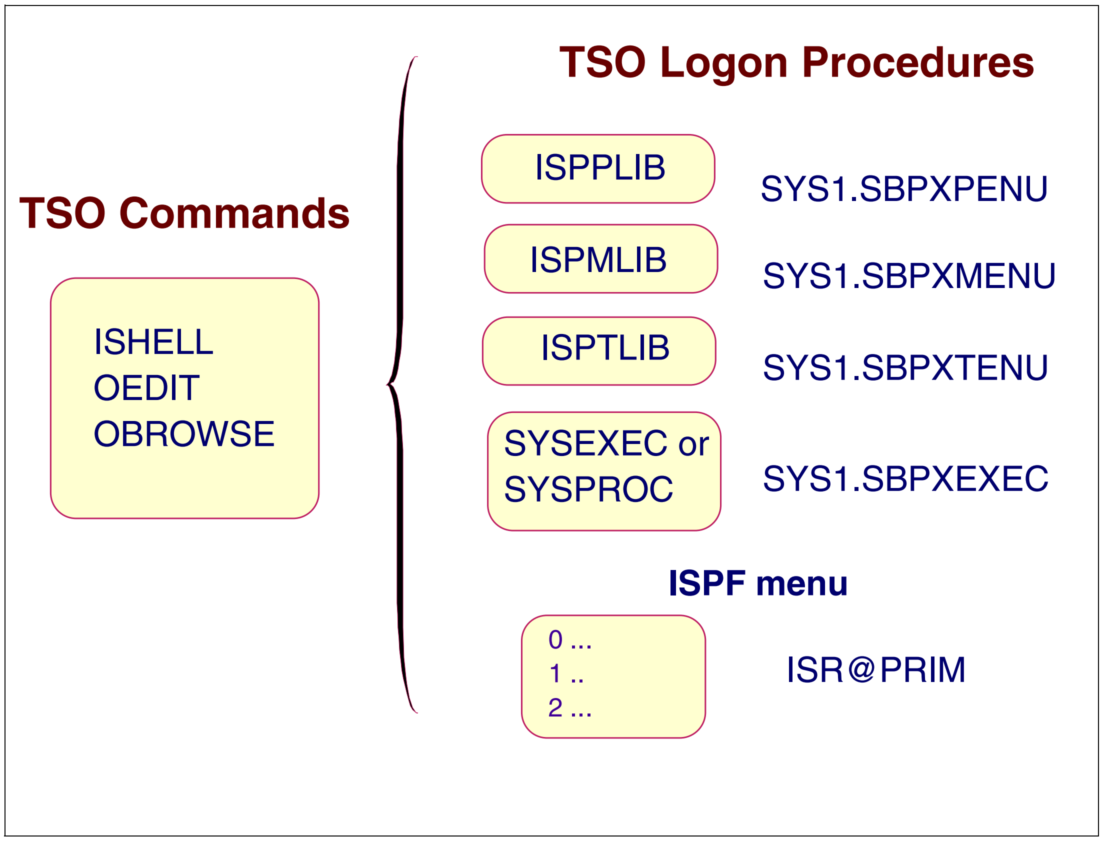


Figure 3-16 Using TSO/E command with z/OS UNIX

In order to make the TSO/E commands OEDIT, OBROWSE, and ISHELL (ISPF Shell) available to users, the following data sets should be concatenated to the appropriate ISPF DD names (in the TSO/E logon procedure, or CLIST/REXX EXEC if used):

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

These TSO/E commands can be invoked from a TSO/E command line, but it is more convenient for users if they are added as an option to an ISPF menu. They can be added to the ISPF/PDF primary option menu (ISR@PRIM) or any other menu the installation prefers to use.

TSO/E users need to be defined with an OMVS segment in RACF before they can use the ISHELL, OEDIT, or OBROWSE commands.

4

UNIX System Services installation

This chapter provides information on installing UNIX System Services, and how to avoid or fix installation problems. It explains how to:

- Install UNIX System Services
- Allocate HFS data sets
- Mount HFS data sets
- Restore file systems

4.1 IEASYSxx parmlib member

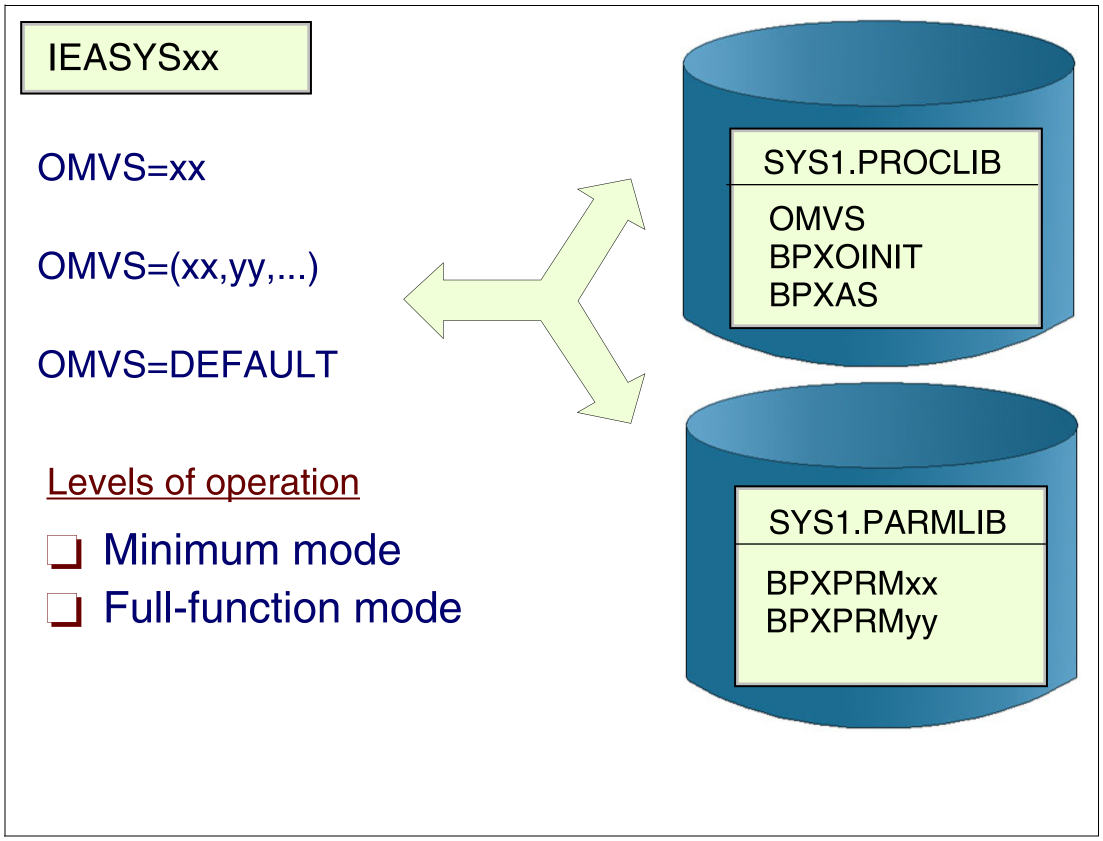


Figure 4-1 Parmlib settings needed to start z/OS UNIX

The initial parmlib settings for the z/OS UNIX kernel are pointed to by the OMVS parameter in the IEASYSxx parmlib member. The OMVS parameter in the IEASYSxx parmlib member lets you specify one or more BPXPRMxx parmlib members to be used to specify the initial parmlib settings for the kernel. If you do not specify the OMVS parameter, or if you specify OMVS=DEFAULT, the kernel is started in a minimum configuration mode with all BPXPRMxx parmlib statements taking their default values.

OMVS may also be left out or coded as DEFAULT. This allows the z/OS UNIX kernel to start in a minimum configuration. All BPXPRMxx values will take their default values and a temporary root file system will be set up in memory.

Note: The start and stop commands for the z/OS UNIX kernel are no longer supported.

Activation of kernel services is available in two modes:

- Minimum mode
- Full-function mode

You can set up kernel services in either 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 will need to use full function mode; otherwise, you can use minimum mode. In order to apply service to the HFS, you need at least one system that can run in full function mode.

4.2 z/OS UNIX minimum mode

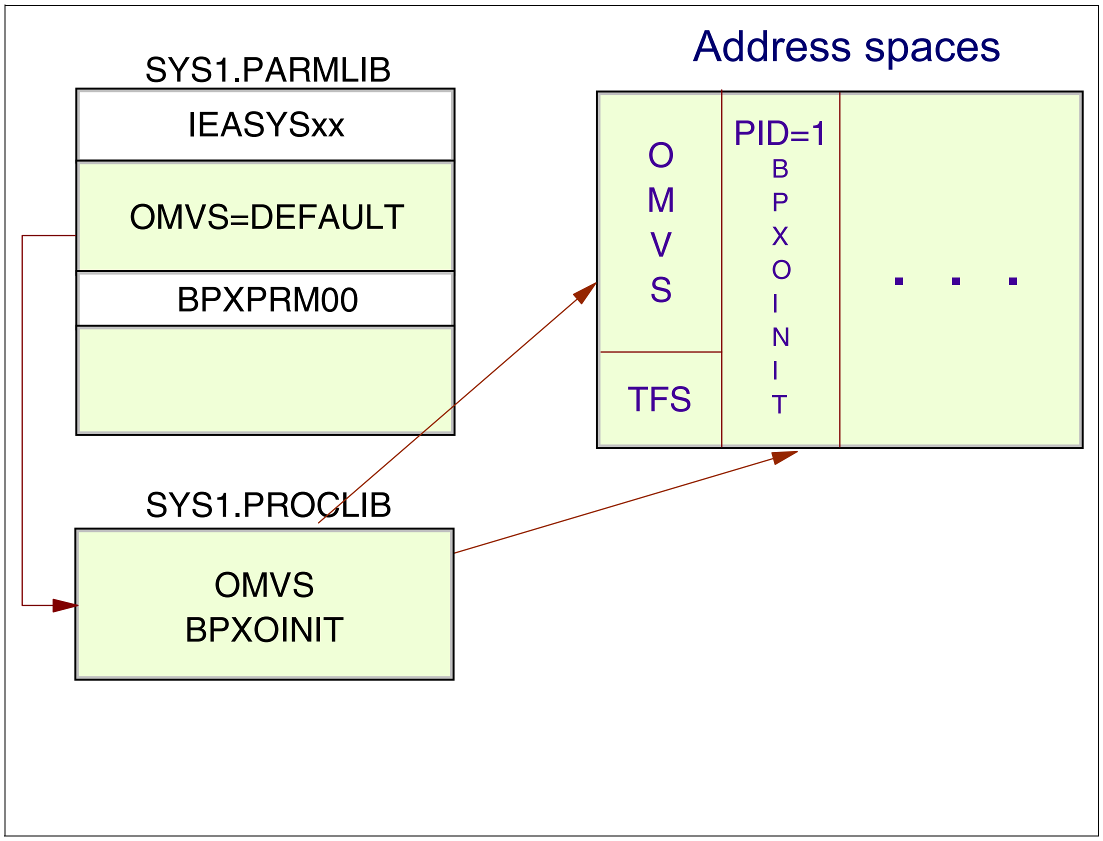


Figure 4-2 z/OS UNIX running in minimum mode

Minimum mode is intended for installations that do not intend to use z/OS UNIX System Services, but which allow the IPL process to complete. In this mode many services are available to programs. Some that require further customization, such as a fork(), will fail.

If you do not specify OMVS= in the IEASYSxx PARMLIB member or if you specify OMVS=DEFAULT, then kernel services start up in minimum mode when the system is IPLed. This mode is intended for installations that do not plan to use the kernel services. In minimum mode:

- Many services are available, but some functions such as TCP/IP sockets that require other system customization may not work.
- TCP/IP sockets (AF_INET) are not available.
- A temporary file system (TFS) is used. A TFS is an in-storage file system, hence no physical DASD data set needs to exist or be mounted.

A temporary file system (kept in memory) is created for the root. The required directories (/bin, /etc, /tmp, /dev, and /dev/null) are built. There are no executables in this file system.

4.3 Minimum mode: TFS

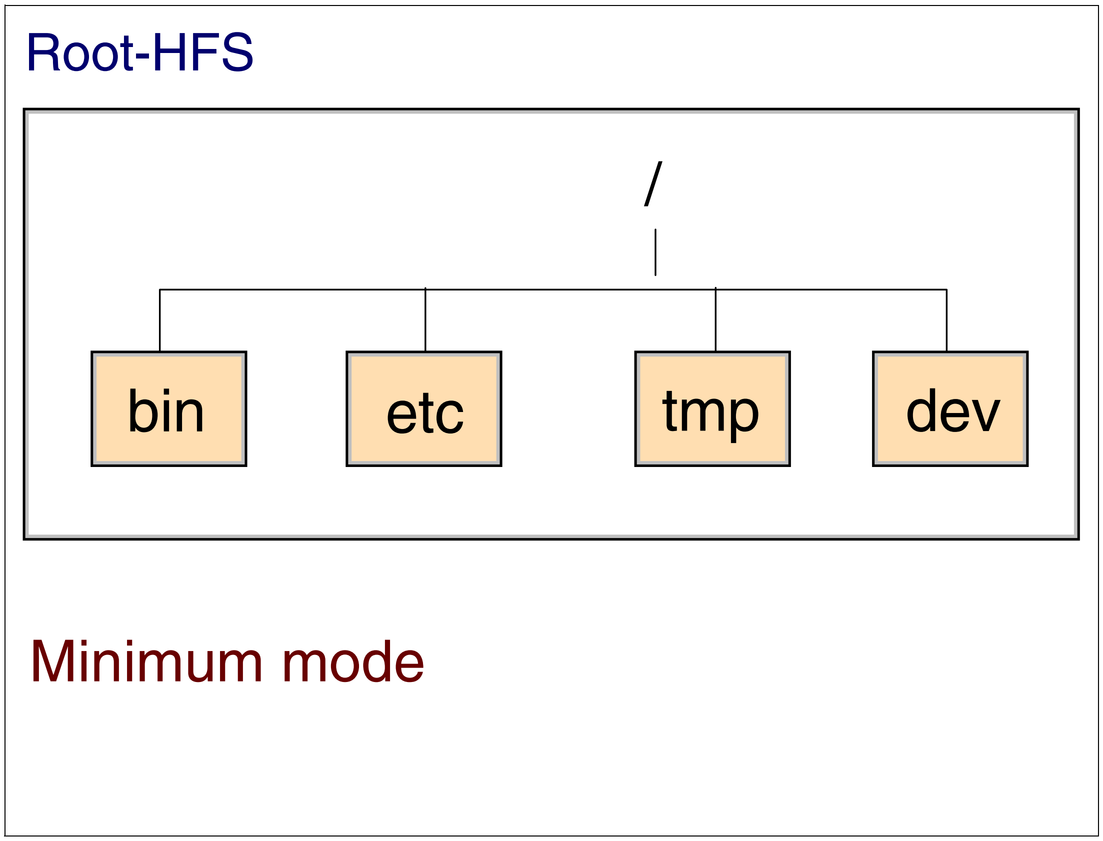


Figure 4-3 The TFS in minimum mode

A temporary file system is an in-memory file system which delivers high-speed I/O. If the kernel is started in minimum setup mode, the TFS is automatically mounted. The system is in minimum mode when:

- OMVS=Default
- There is no OMVS Statement in IEASYSxx (no BPXPRMxx member in the PARMLIB)

A temporary file system is used as the root file system. The temporary file system is initialized and primed with a minimum set of files and directories, specifically the following:

```
/ (root directory)
/bin directory
/etc directory
/tmp directory
/dev directory
/dev/null file
```

Note: Any data written to this file system is not written to DASD.

There are no executables in the temporary file system (that is, you will not find the Shell and Utilities). Do not attempt to install z/OS UNIX System Services Application Services in the TFS, since no data will be written to DASD. Because the TFS is a temporary file system, unmounting it causes all data stored in the file system to be discarded. If, after an unmount, you mount another TFS, that file system has only a dot (.) and a dot-dot (..) and nothing else.

4.4 z/OS UNIX full-function mode

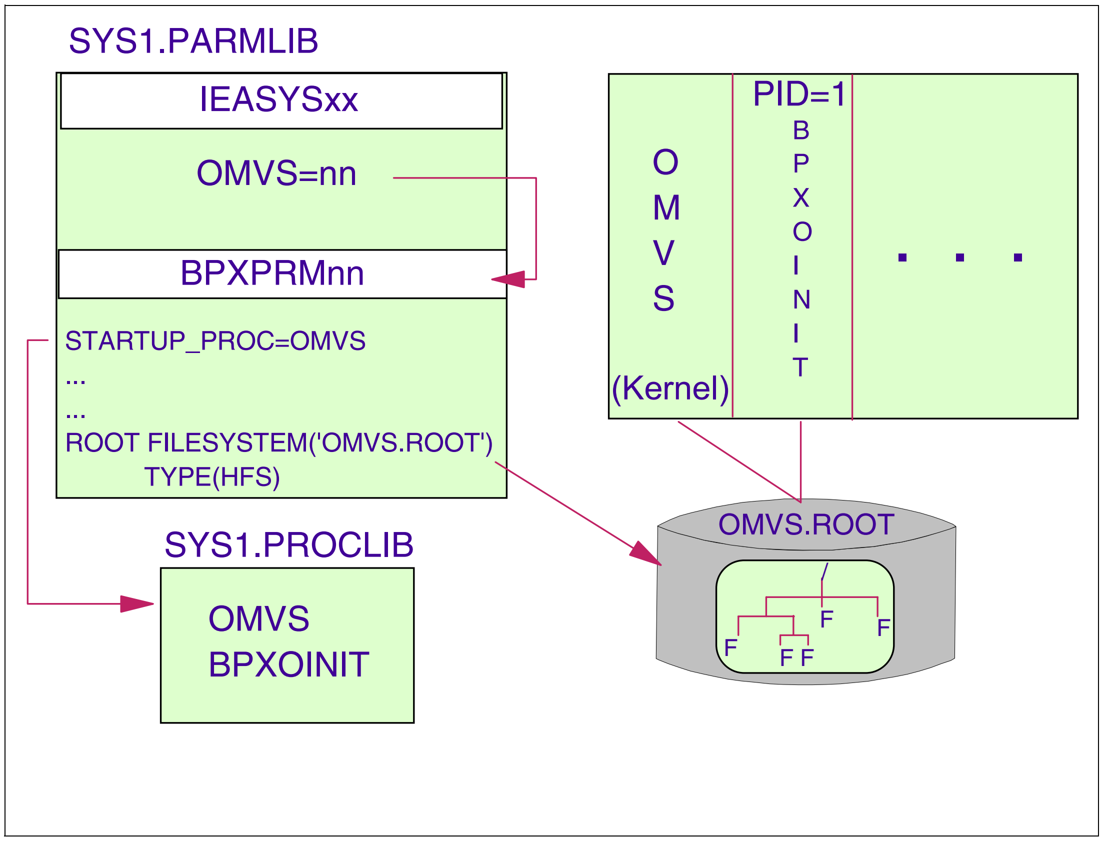


Figure 4-4 z/OS UNIX full function mode

Full-function mode is started at IPL time when the OMVS parameter in the IEASYSxx parmlib member points to one or more BPXPRMxx parmlib members.

There must be a root HFS built and defined in BPXPRMxx for OMVS to initialize correctly, and SMS, WLM, and RACF customization should be completed.

BPXPRMnn

STARTUP_PROC is a 1 to 8 character name of a started procedure JCL that initializes the z/OS UNIX kernel. The default is OMVS.

The ROOT statement defines and mounts the root file system. In this example:

- HFS is the TYPE of the file system.
- OMVS.ROOT is the file system, which is the name of an already defined HFS data set.
- The root file system has a default of read/write mode.

If an active BPXPRMxx PARMLIB member specifies "`FILESYSTEM TYPE(HFS)`", then the kernel services start up in full-function mode when the system is IPLed. To use the full function, you need to:

- Set up BPXPRMxx PARMLIB definitions
- Set up DFSMS
- Set up the hierarchical file systems
- Set up the security product definitions for z/OS UNIX
- Set up the users' access and their individual file systems

DFSMS manages the HFS data sets that contain the file systems.

Note: APAR OW35441 now gives you the ability to allocate PDSE and HFS data sets on unmanaged (non-SMS) volumes, if running DFSMS 1.4 or DFSMS 1.5.

BPXOINIT

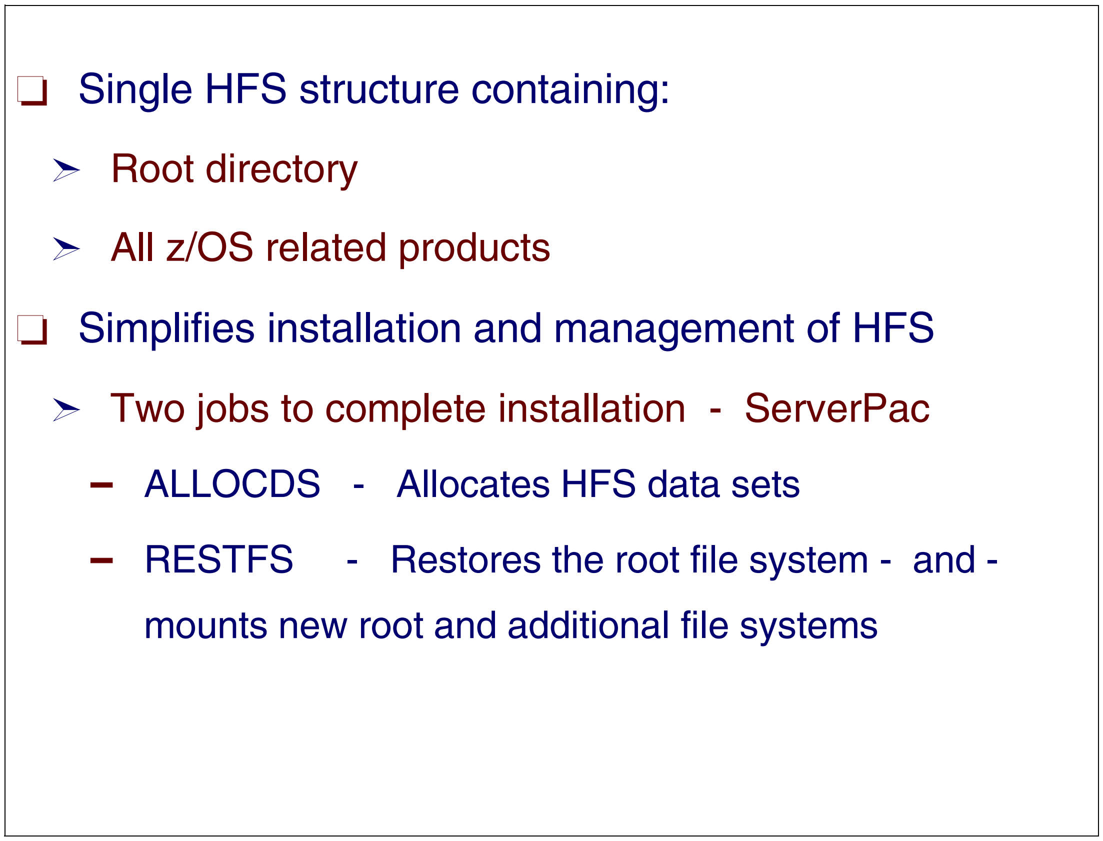
As of OS/390 V2R3, BPXOINIT is the started procedure that runs the initialization process. BPXOINIT is also the jobname of the initialization process. Prior to OS/390 V2R3, the initialization process was created via an APPC allocate and the jobname was OMVSINIT. BPXOINIT is shipped in SYS1.PROCLIB.

At system IPL time, kernel services are started automatically. If the OMVS parameter in the IEASYSxx parmlib parameter is not specified, the kernel services are started in minimum mode. If the OMVS parameter specifies one or more BPXPRMxx parmlib members, they are all used to configure the kernel services when the system is IPLed.

The BPXOINIT address space has two categories of functions:

1. It behaves as PID(1) of a typical UNIX system. This is the parent of /etc/rc, and it inherits orphaned children so that their processes get cleaned up using normal code in the kernel. This task is also the parent of any MVS address space that is dubbed and not created by fork or spawn. Therefore, TSO/E commands, batch jobs, and so forth have a parent PID of 1.
2. Certain functions that the kernel performs need to be able to make normal kernel calls. This address space is used for these activities, for example, mmap() and user ID alias processing.

4.5 z/OS HFS root



- ❑ Single HFS structure containing:
  - ➢ Root directory
  - ➢ All z/OS related products
- ❑ Simplifies installation and management of HFS
  - ➢ Two jobs to complete installation - ServerPac
    - – ALLOCDS - Allocates HFS data sets
    - – RESTFS - Restores the root file system - and - mounts new root and additional file systems

Figure 4-5 z/OS UNIX root install using ServerPac

The z/OS ServerPac provides two jobs to complete the installation of the root HFS. They create a single HFS data set structure that contains the following:

- ► The root directory
- ► All z/OS related products placed into the ServerPac order that require UNIX System Services.

ALLOCDS job

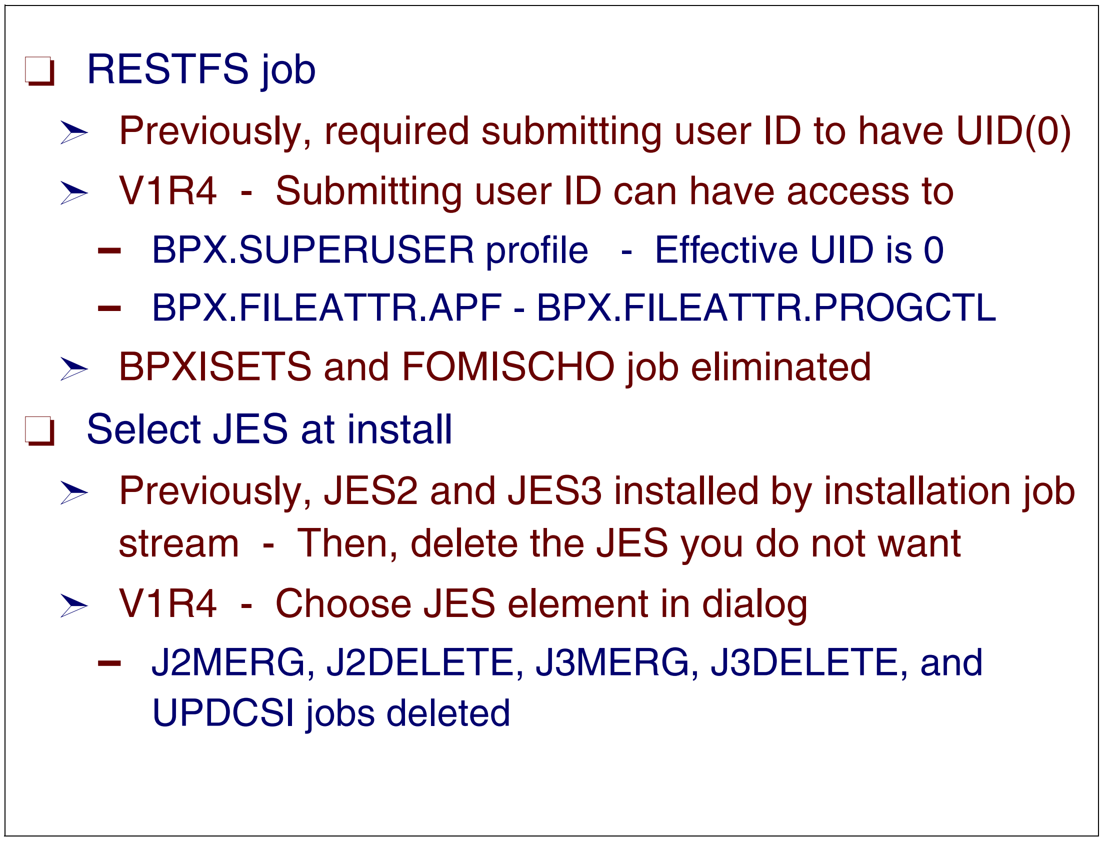
This job allocates HFS data sets.

RESTFS job

This job mounts the new Root file system, and creates mount points for and mounts the additional file systems. The BPXPRMFS member of CPAC.PARMLIB is updated to reflect the file system structure.

The RESTFS job then restores those files into the Root file system.

4.6 Installation improvements (ServerPac)



- ❑ RESTFS job
  - ➢ Previously, required submitting user ID to have UID(0)
  - ➢ V1R4 - Submitting user ID can have access to
    - – BPX.SUPERUSER profile - Effective UID is 0
    - – BPX.FILEATTR.APF - BPX.FILEATTR.PROGCTL
  - ➢ BPXISETS and FOMISCHO job eliminated
- ❑ Select JES at install
  - ➢ Previously, JES2 and JES3 installed by installation job stream - Then, delete the JES you do not want
  - ➢ V1R4 - Choose JES element in dialog
    - – J2MERG, J2DELETE, J3MERG, J3DELETE, and UPDCSI jobs deleted

Figure 4-6 ServerPac enhancements in z/OS V1R4

The RESTFS job (and corresponding subsystem jobs) previously required the submitting user ID to have UID(0) set in its OMVS segment. Access to the BPX.SUPERUSER profile in the FACILITY class would provide the required authority without the need for UID(0). The ServerPac for z/OS V1R4 now sets the effective UID to zero when the user ID has access to BPX.SUPERUSER. This eliminates the need for the user executing the restore of the ServerPac to have UID(0) authority.

In addition, regardless of your installation method, the user ID must be permitted read access to the FACILITY classes BPX.FILEATTR.APF and BPX.FILEATTR.PROGCTL (or BPX.FILEATTR.* if you choose to use a generic facility for both facility classes).

In order to define these FACILITY classes, you can use the following commands (which are also provided in SYS1.SAMPLIB):

```
RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
RDEFINE FACILITY BPX.FILEATTR.PROGCTL UACC(NONE)
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY)
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
```

Note: This does not affect the pax utility. You still required UID(0) in order to execute the pax command outside of the ServerPac processing. This change is for ServerPac processing only.

RESTFS is historically the number one source of service calls. To help eliminate some of the problems, logic checking for common setup problems, new error messages, and more specific error messages for better diagnosis have been added. RESTFS calls pax and BPXISETS from within the exec to eliminate bypass checking and job elimination. A new status message indicator shows progress of the job.

BPXISETS job eliminated

The RESTFS changes have eliminated the need to run BPXISETS from the ServerPac dialogs. An element change has removed the need to run the FOMISCHO job.

Select JES at install

Previously, both JES2 and JES3 were installed by running the installation job stream. The customer would then delete the JES element not wanted, and then optionally merge the JES used with the BCP zones and reallocate the DASD space used by the deleted element. With the z/OS V1R4 ServerPac, the customer can choose either JES element, or both, in the dialog and specify whether they are to be merged. Then the installation job stream is run.

The new ServerPac function of JES selection has removed the J2MERG, J2DELETE, J3MERG, J3DELETE, and UPDCSI jobs.

4.7 Current HFS root structure

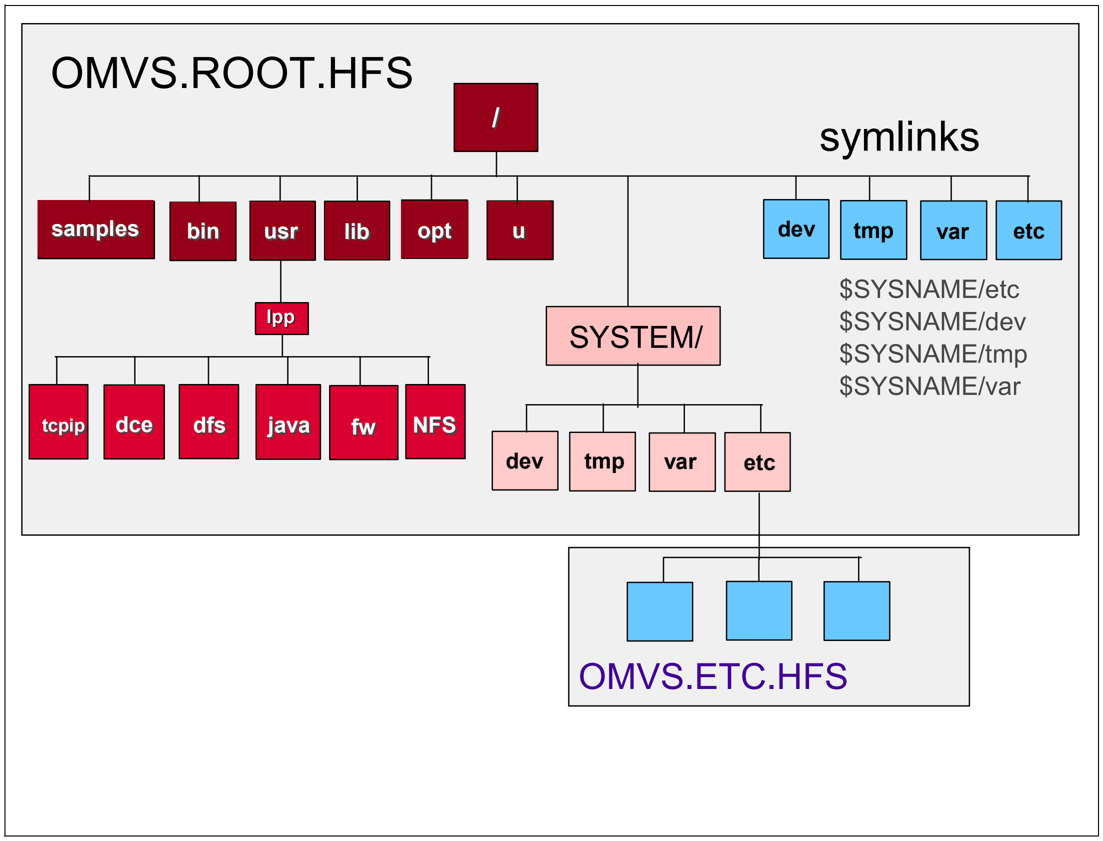


Figure 4-7 z/OS UNIX root file structure

Figure 4-7 shows the root file structure that began with OS/390 V2R9. For ServerPac customers, IBM delivers a single-root HFS. This HFS is unloaded when you do what is called “Establishing UNIX Services” in the ServerPac installation process.

Not only does the single-root HFS make cloning of file systems easier, but it also dramatically reduces the number of jobs run by system programmers to establish UNIX System Services. The ServerPac recommends that **/etc** be mounted separately and supplies a job to do this.

After completing the two install jobs, ALLOCDS and RESTFS, there will be an HFS mounted to the /etc directory. Keeping the /etc file system in an HFS data set separate from other file systems allows you to separate your customization data from IBM's service updates, and also eases migration to another release. The install process creates an empty /etc directory into which customers must copy their existing /etc file system.

z/OS UNIX files are organized in a hierarchical file system, as in other UNIX systems. 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. The figure shows the root and the directories that exist when the system is installed. Under the /usr/lpp directory are directories for z/OS elements and features.

If you follow the instructions for ServerPac and CBPDO installations, all z/OS elements and features that store into the HFS are installed into a consolidated HFS data set, instead of having separate product-related HFS data sets. This began with OS/390 V2R6.

We recommend that you continue this consolidated approach as you install additional products on the platform. This makes maintaining and cloning the file system easier, and it simplifies the MOUNT statements in the BPXPRMxx parmlib member.

IBM requires that you maintain a separate HFS data set for each of the following directories:

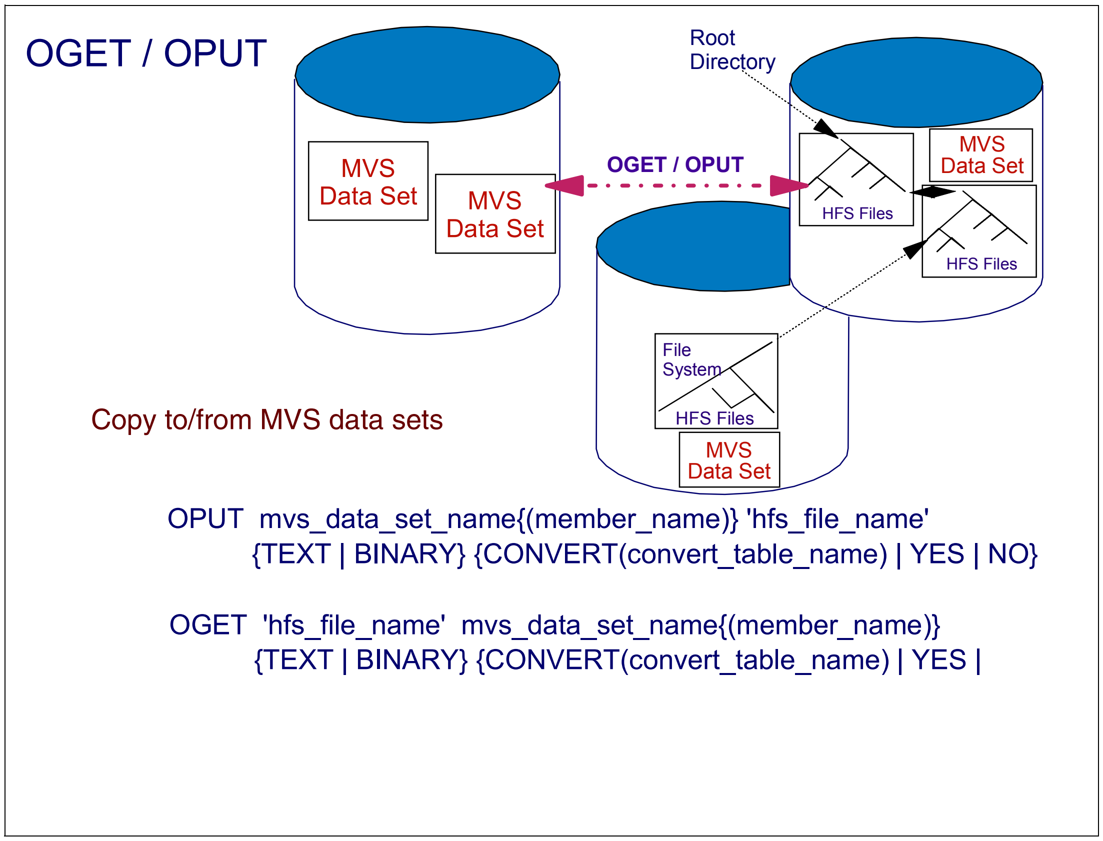
/etc /etc contains customization data. Keeping the /etc file system in an HFS data set separate from other file systems allows you to separate your customization data from IBM's service updates. It also makes migrating to another release easier. After you complete instructions for a ServerPac or CBPDO installation, you will have an /etc file system in its own HFS data set.

/dev /dev contains character-special files that are used when logging into the OMVS shell environment and also during c89 processing. Prior to OS/390 V2R7, these character-special files were created by running the BPXISMKD REXX exec or would be part of your ServerPac order. Beginning with OS/390 V2R7, /dev is shipped empty. The necessary files are created when the system is IPLed, and on a per-demand basis.

/tmp /tmp 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.

/var /var 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 during execution of code. An example of this is caching data. In addition, you can be assured that IBM products will only create directories or files when code is executed.

4.8 UNIX utilities: TSO/E commands



```
OPUT  mvs_data_set_name{(member_name)} 'hfs_file_name'
        {TEXT | BINARY} {CONVERT(convert_table_name) | YES | NO}

OGET  'hfs_file_name'  mvs_data_set_name{(member_name)}
        {TEXT | BINARY} {CONVERT(convert_table_name) | YES |
```

Figure 4-8 TSO/E OGET and OPUT commands

You can use the **OGET** command to copy an HFS file:

- To a member of an MVS partitioned data set (PDS or PDSE)
- To an MVS sequential data set

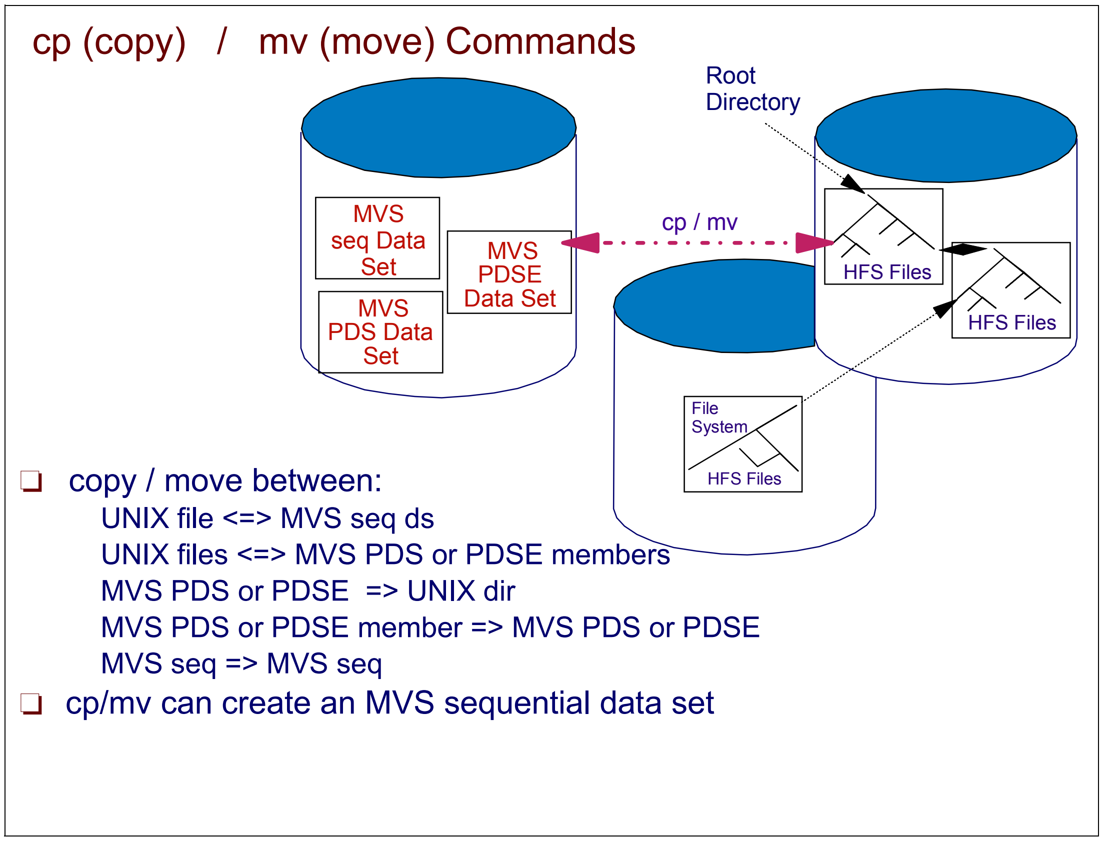
You can also convert the data from code page 1047 to code page IBM-037 or ASCII while it is being copied. Do not use the CONVERT option when copying files that contain doublebyte data. This option is used for singlebyte data only, not for doublebyte data.

You can use the **OPUT** command to:

- Copy a member of an MVS partitioned data set (PDS or PDSE) to a file
- Copy an MVS sequential data set to a file

You can also convert the data from code page IBM-037 or ASCII to code page IBM-1047.

4.9 Updated UNIX commands: OS/390 V2R8



cp (copy) / mv (move) Commands

- copy / move between:
  - UNIX file <=> MVS seq ds
  - UNIX files <=> MVS PDS or PDSE members
  - MVS PDS or PDSE => UNIX dir
  - MVS PDS or PDSE member => MVS PDS or PDSE
  - MVS seq => MVS seq
- cp/mv can create an MVS sequential data set

Figure 4-9 New updates to commands with OS/390 V2R8

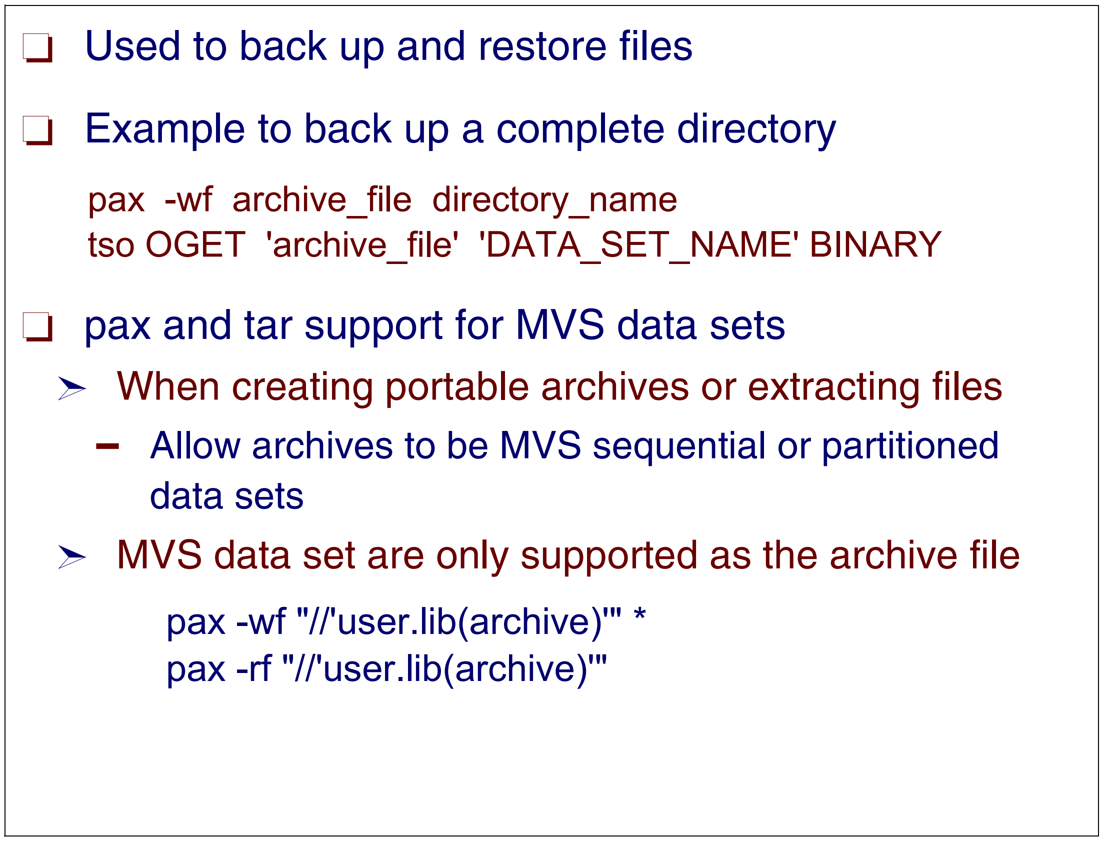
You can use **cp** and **mv** UNIX commands to copy or move files to and from MVS data sets. If you specify more than one file to be copied, the target (last pathname on the command line) must be either a directory or a partitioned data set. If the target is an MVS partitioned data set, the source cannot be a UNIX directory.

You can now copy or move:

- A UNIX file to an MVS data set
- An MVS data set to a filesystem
- An MVS data set to an MVS data set

To **CP** or **MV** to a PDS or PDSE, the data set must be allocated before the copy or move. You may *not* **cp/mv** a partitioned data set to another partitioned data set and you may *not* **cp/mv** a directory to a partitioned data set (you may use **dir /*** to accomplish copying/moving all files from the directory to a partitioned data set).

4.10 pax and tar utilities



- ❑ Used to back up and restore files
- ❑ Example to back up a complete directory

```
pax  -wf  archive_file  directory_name
tso OGET  'archive_file'  'DATA_SET_NAME' BINARY
```

- ❑ pax and tar support for MVS data sets
  - ➢ When creating portable archives or extracting files
    - – Allow archives to be MVS sequential or partitioned data sets
  - ➢ MVS data set are only supported as the archive file

```
pax -wf "//'user.lib(archive)'" *
pax -rf "//'user.lib(archive)'"
```

Figure 4-10 pax and tar utilities

Use **pax** to read, write and list archive files. An archive file is a single file containing one or more files and directories. Archive files can be HFS files or MVS data sets. A file stored inside an archive is called a component file; similarly, a directory stored inside an archive is called a component directory.

Note: MVS data sets cannot be specified for component files. Included with each component file and directory is recorded information such as owner and group name, permission bits, file attributes, and modification time.

You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

Archives created by pax are interchangeable with those created with the **tar** utility. Both utilities can read and create archives in the default format of the other (USTAR for pax and TAR for tar). Archives are generally named with suffixes such as .pax or .tar (or pax.Z and tar.Z for compressed files), but this is not required.

pax utility

The pax utility can read and write files in CPIO ASCII format, CPIO binary format, TAR format, or USTAR format. It can read files that were written using tar, cpio, or pax itself. How it handles filename length and preservation of link information across the backup and restore process depends on the format you select: If you select CPIO, it behaves like the **cpio** command; and if you select TAR, it behaves like the **tar** command.

Figure 4-10 shows a pax example to back up a complete directory, including the subdirectories and their contents, into a data set.

In the example:

directory_name	is the name of the directory you want to archive
archive_file	is an absolute pathname
DATA_SET_NAME	is a fully qualified data set name

The `pax` command creates an archive file with the specified name in the current working directory.

The `OGET` command copies the archive file into the specified MVS data set.

OS/390 V2R8 enhancements

With Release 8 and later, the pax and tar utilities can read an archive file directly from an MVS data set. Use the pax or tar shell command to restore the directory or file system from the archive file; all the component files are restored from the archive file.

pax and tar package HFS directory trees into a single file called an "archive." Archives are always treated as binary.

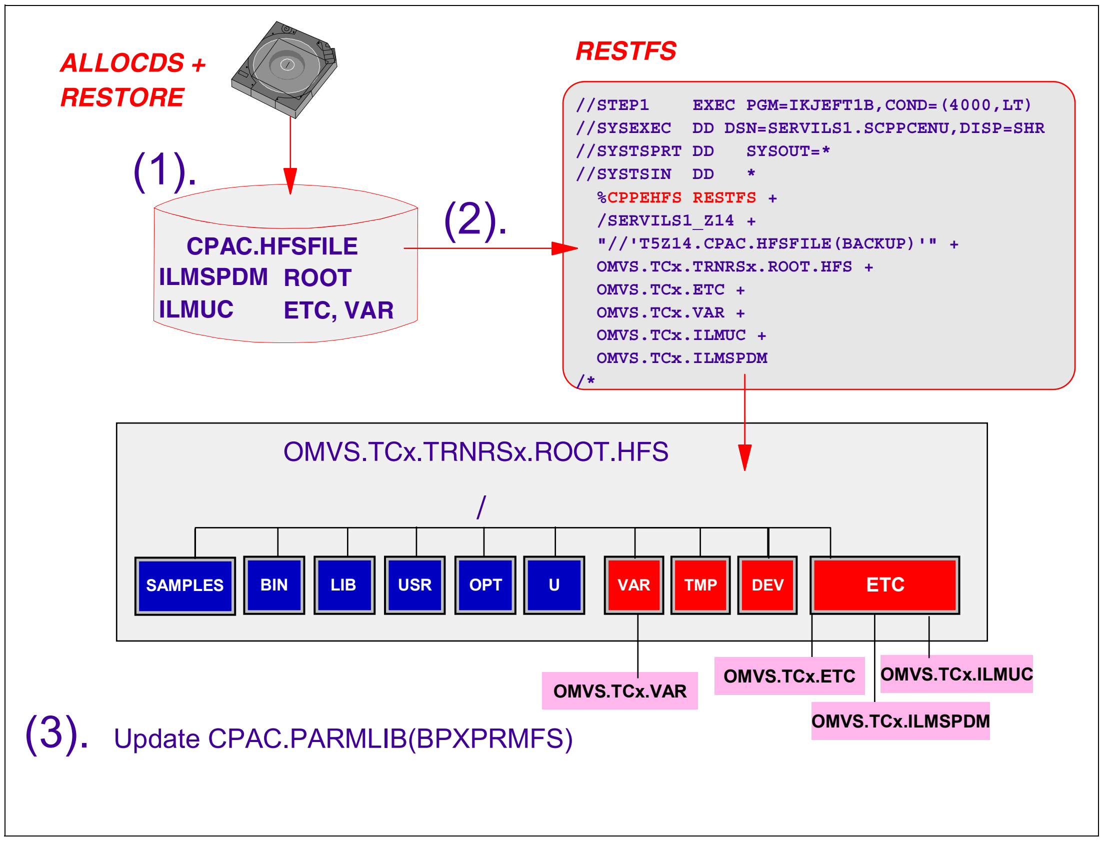
Only partitioned data sets in undefined format may store an executable.

Archives can be moved to other systems or directories.

"Extracting files" means moving files from the archive into HFS files. This avoids OPUT of archives from MVS data sets to UNIX, and `OGET` of archives from UNIX to MVS data sets.

Note: When writing to a PDS member, the PDS must already exist; pax will automatically overwrite an existing archive.

4.11 ServerPac z/OS UNIX installation



```
//STEP1     EXEC PGM=IKJEFT1B,COND=(4000,LT)
//SYSEXEC   DD DSN=SERVILS1.SCPPCENU,DISP=SHR
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
  %CPPEHFS RESTFS +
  /SERVILS1_Z14 +
  "//'T5Z14.CPAC.HFSFILE(BACKUP)'" +
  OMVS.TCx.TRNRSx.ROOT.HFS +
  OMVS.TCx.ETC +
  OMVS.TCx.VAR +
  OMVS.TCx.ILMUC +
  OMVS.TCx.ILMSPDM
/*
```

Figure 4-11 ServerPac flow for z/OS UNIX installation

For a full configuration you need to build a complete root file system.

ServerPac supplies the jobs, ALLOCDS and RESTORE, to download the PDS from tape to DASD. The other job, RESTFS uses the pax utility to restore the HFS file systems.

After running these jobs you have a complete root file system containing all the root-level directories, files, and programs.

ServerPac builds the root for you with all the features that you ordered already installed in it. They use the UNIX pax utility to compress the hierarchical format into an HFS file. It is distributed to you as a member of hlq.HFSFILE:

```
hlq.HFSFILE(BACKUP)  Backup of Root File System
```

In addition to reading and writing archive files (which concatenate the contents of files and directories), pax can record file modification information such as dates, owner names, and so on. You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

(1) ALLOCDS

The ServerPac job ALLOCDS performs the following tasks for you:

- Creates ROOT HFS
- Creates ETC HFS

- Creates the following directories:

  ILMSPDM - HFS
  ILMUC - HFS
  VAR - HFS

(2) RESTFS job

The next job to run is RESTFS. This job does the following tasks for you:

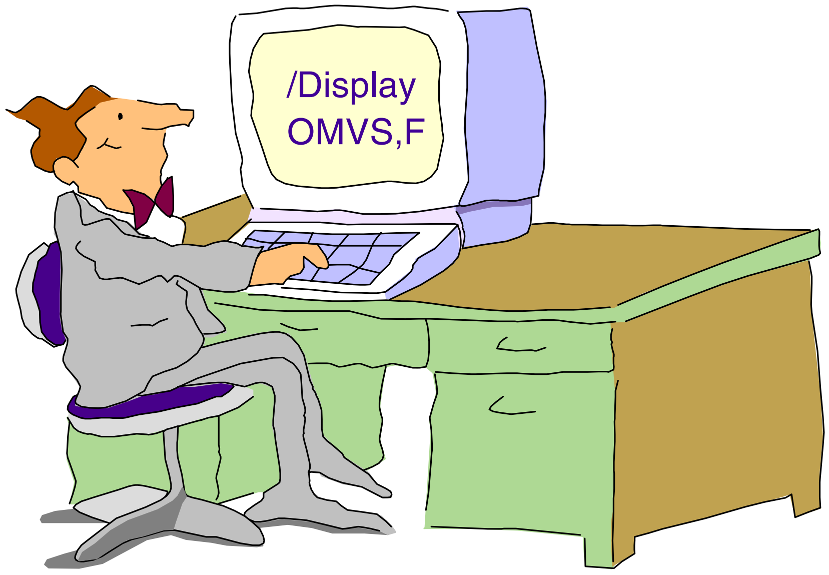
- Allocates STDIN, STDOUT and STDERR files (UNIX standard outputs). An RC=4 is received if the directory is empty.
- Creates **/SERVICE** directory.
- Calls `pax` to restore the ROOT HFS.
- Mounts the created ROOT HFS to **/SERVICE**.
- Mounts the created ETC HFS to **/etc** and mounts var, ilmuc, and ilmspdm HFS data sets.

(3) Update parmlib

The installation process RESTFS job updates the CPAC.PARMLIB member as follows:

- Updates the BPXPRMFS with the MOUNT FILESYSTYPE section
- Updates the BPXPRMFS with the FILESYSTYPE TYPE(....) and the corresponding entry points

4.12 UNIX System Services installation



```
BPXO044I 15.23.48 DISPLAY OMVS 971
OMVS      000E ACTIVE          OMVS=00
TYPENAME   DEVICE ----------STATUS----------- MODE QJOBNAME  QPID
BPXTFS          1 ACTIVE                      RDWR
  NAME=ROOT
  PATH=/
```

Figure 4-12 Display of root file system

Before we update the provided JCL we should check how the TFS was named. It should be equal to the name we used in our first step, where we requested an unmount for the TFS.

You can get the information using the **D OMVS,F** command.

This command provides the following output:

- BPXTFS NAME=ROOT
- PATH = /
- UNIX Service is ACTIVE
- HFS is in Read/Write (RDWR) Mode

The TFS is used to get UNIX System Service up and running during IPL. Since OS/390 V2R3 it is no longer possible to start and stop UNIX System Services from OMVS.

Using the **D OMVS,F** command will show the mounted file systems; this can also be done by using the UNIX ISHELL.

5

z/OS UNIX shell and utilities

This chapter provides information on how to invoke, customize, and use the z/OS UNIX shell and utilities. It includes details on the following topics:

- A brief overview about the z/OS UNIX shell
- How to invoke the shell using different methods
- Creating the required resources for the shell
- Setting up code page translation for the shell
- Shell environment variables
- Global variables
- How to pick up the region size when invoking the shell
- Where and how to set up the time zone variable
- Customizing the c89/cc compiler
- Installing books for OHELP

5.1 The z/OS UNIX shell

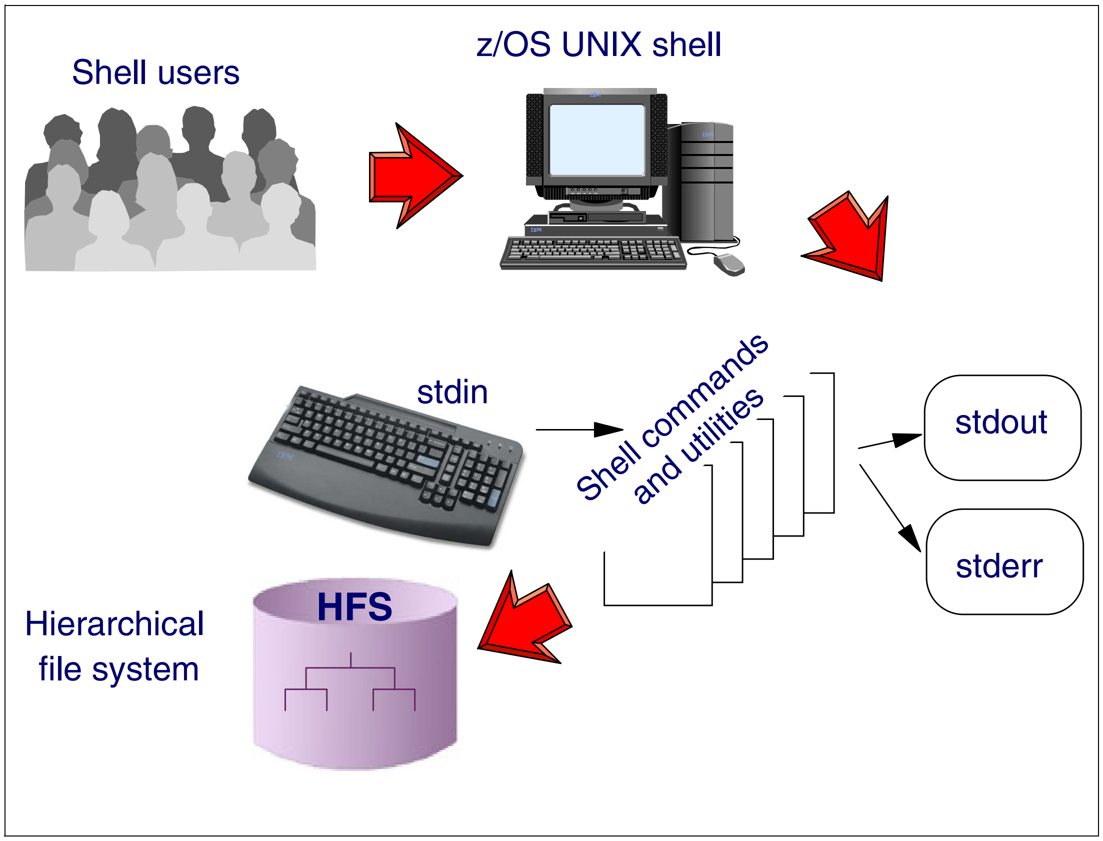


Figure 5-1 Overview of the z/OS UNIX shell

The z/OS UNIX shell can be compared to TSO/E for z/OS. It is the interactive interface to z/OS UNIX. The shell feature comes with multiple commands and utilities which are in most cases the same as the ones that come with a UNIX system. The z/OS UNIX shell is based on the UNIX System V shell with some of the features from the Korn shell. The z/OS UNIX shell conforms to the POSIX 1003.2 standard and to the XPG4 standard.

POSIX 1003.2 distinguishes between a command (a directive to the shell to perform a specific task) and a utility (a program callable by name from the shell). To the end user there is no difference between a command and a utility. For the purpose of this discussion, command refers to both commands and utilities.

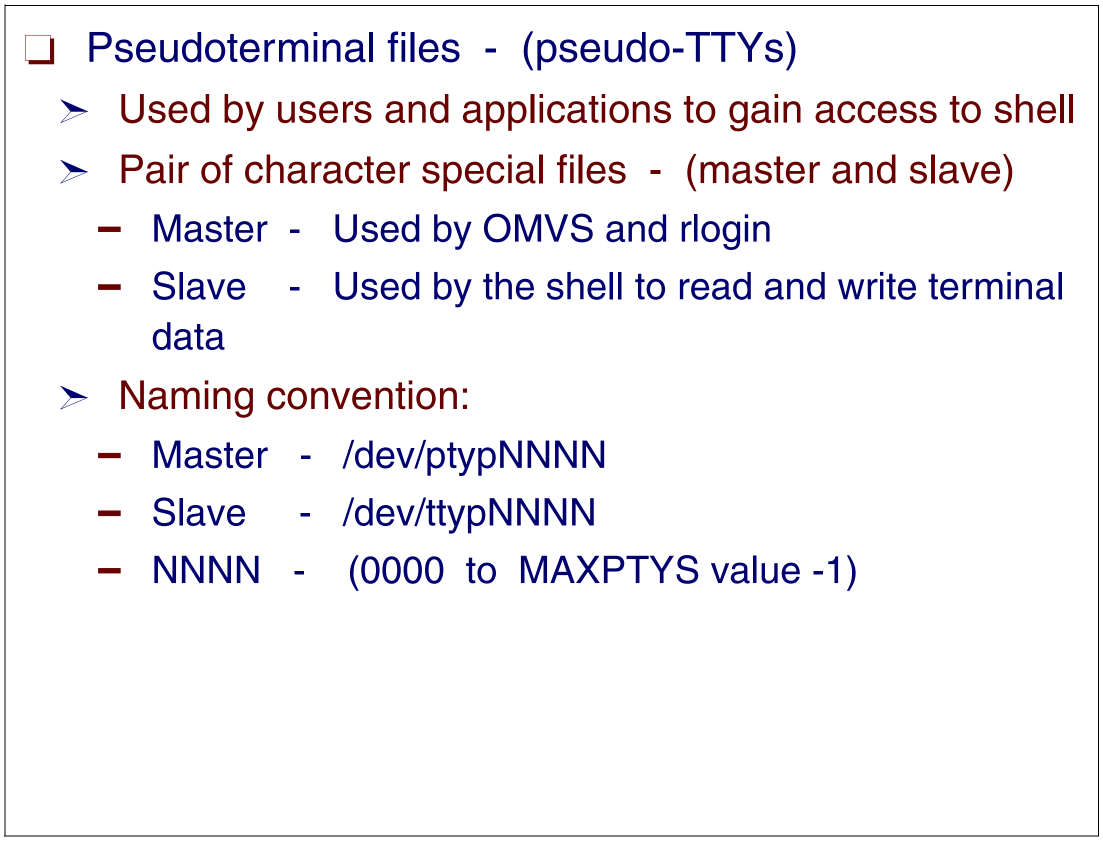
The shell is a command processor that can be used to:

- Invoke shell commands or utilities that request services from the system.
- Write shell scripts using the shell programming language.
- Run shell scripts, REXX EXECS, and C-language programs interactively, in the background, or in batch.

Shell commands are typically short, and they can be combined in pipes, or in shell scripts. Once a shell command begins executing it has access to three files:

```
stdin:    standard input    Default is the keyboard.
stdout:   standard output   Default is the screen.
stderr:   standard error    Default is the screen.
```

5.2 Accessing the z/OS UNIX shell



- ❑ Pseudoterminal files - (pseudo-TTYs)
  - ➢ Used by users and applications to gain access to shell
  - ➢ Pair of character special files - (master and slave)
    - Master - Used by OMVS and rlogin
    - Slave - Used by the shell to read and write terminal data
  - ➢ Naming convention:
    - Master - /dev/ptypNNNN
    - Slave - /dev/ttypNNNN
    - NNNN - (0000 to MAXPTYS value -1)

Figure 5-2 Files used when accessing the z/OS UNIX shell

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 master file and a corresponding slave file. The master file is used by a networking application such as OMVS or rlogin. The corresponding slave 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:

```
/dev/ptypNNNN for the master (major 1)
/dev/ttypNNNN for the slave (major 2)
```

NNNN is between 0000 and one less than the MAXPTYS value in the BPXPRMxx parmlib member.

When a user enters the TSO/E **OMVS** command 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.

5.3 Controlling session resources

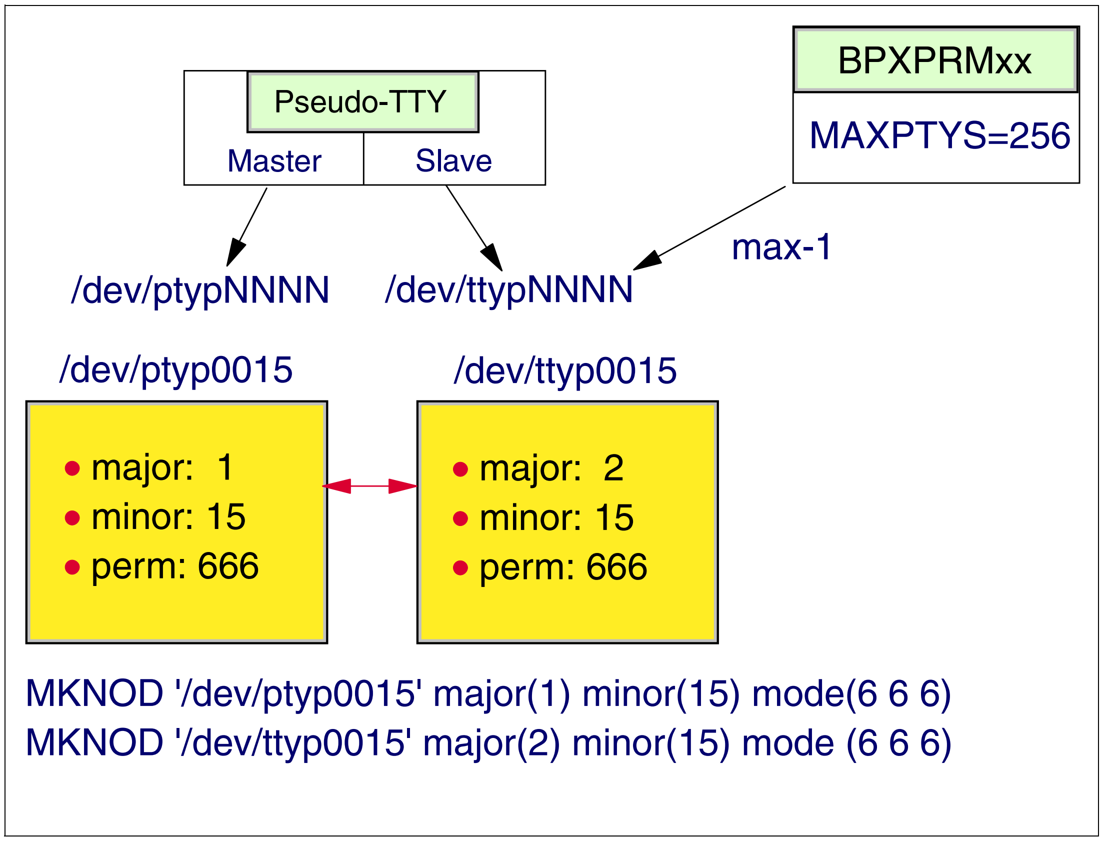


Figure 5-3 Pseudo-TTY files used by shell users

Pseudo-TTY files are dynamically created by the system when they are first referenced. You can add pseudo-TTYs with MKNOD TSO/E commands, as shown in the figure, or with **`mknod`** shell commands. You can also use the ISPF shell to perform these functions.

When using a MKNOD command, make:

- The major number 1 for the master and 2 for the slave
- The minor number the same as the NNNN

The commands can be in a CLIST or REXX exec, or they can be entered directly in a TSO/E session or a shell session.

The MAXPTYS statement specifies the maximum number of pseudo-TTY sessions that can be active at the same time. The range is 1 to 10000; the default and the value in BPXPRMXX is 256.

MAXPTYS lets you manage the number of interactive shell sessions. When you specify this value, each interactive session requires one pseudo-TTY pair. You should avoid specifying an arbitrarily high value for MAXPTYS. However, because each interactive user may have more than one session, it is recommended that you allow four pseudo-TTY pairs for each user (MAXIUDS * 4). The MAXPTYS value influences the number of pseudo-TTY pairs that can be defined in the file system.

5.4 Dynamic /dev

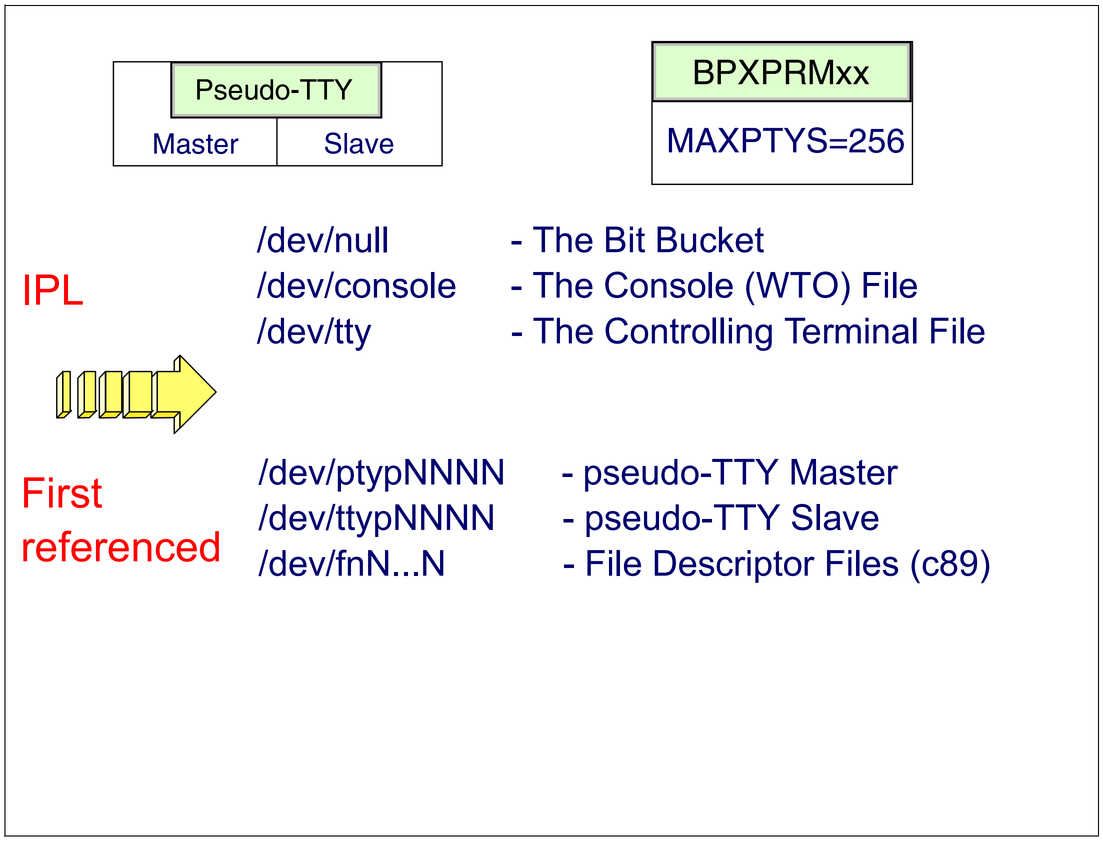


Figure 5-4 Dynamic /dev definitions

The null file, /dev/null, (major 4) 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.

The default controlling terminal can be accessed through the /dev/tty special file (major 3). This file will be defined the first time the system is IPLed.

As part of the ongoing effort to reduce or eliminate post-install steps, the creation of the standard /dev files has been made automatic beginning with OS/390 V2R7. These files were previously created by SYS1.SAMPLIB(BPXMKDIR).

Any reference to pathnames of these formats will cause the corresponding type of character special file to be implicitly defined if it doesn't already exist. This is usually due to open() or stat().

No special authority is needed to dynamically create these files, even if the user would not normally be permitted to define files in the /dev directory. The files are not removed by the system after they have been defined.

5.5 Invoking the shell via TSO/E

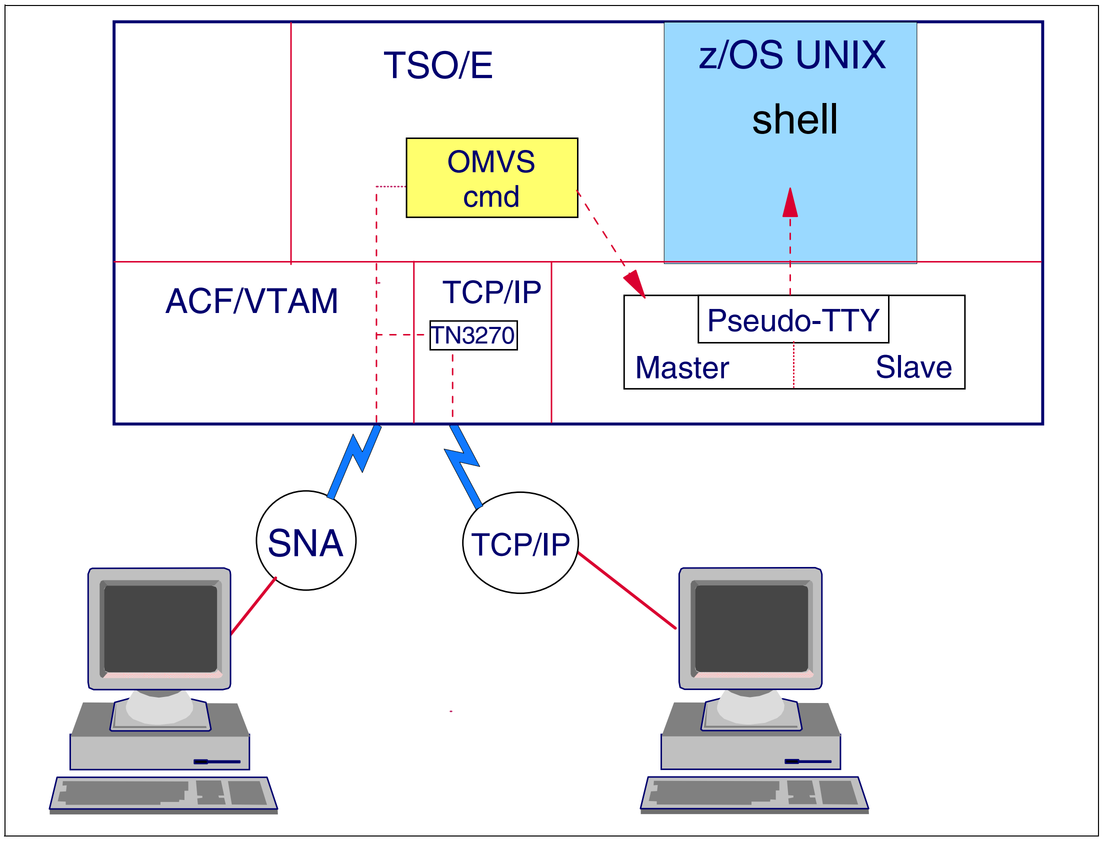


Figure 5-5 Invoking the shell from a remote terminal

One of the methods to access the shell is by logging on to TSO/E and issuing a command called **OMVS**. The network connection to the shell via TSO/E can be either VTAM or TCP/IP. This includes:

- Real and emulated 3270 terminals in an SNA network.
- UNIX systems and other workstations that in a TCP/IP network support the Telnet 3270 (TN3270) client function. The TN3270 client communicates with the Telnet server (TN-S) in TCP/IP on z/OS.

When accessing the shell from TSO/E, the users can jump back and forth between the shell and TSO/E. It is also possible to issue TSO/E commands from the shell.

The **OMVS** TSO/E command, together with the pseudo-TTY function, maps and transforms the 3270-oriented TSO/E terminal interface and user externals to the POSIX 1003.1 defined terminal interface expected by the POSIX 1003.2 conforming shell.

Pseudo-terminals are defined as a pair of character special files, one file designated as the master, and one file as the slave. The pseudo-TTY master process is traditionally a network server application; in z/OS UNIX the TSO/E command processor is the server application. The pseudo-TTY slave process is the user shell process.

The pseudo-TTY function resides in the z/OS UNIX kernel and consists of a master pty and a slave pty. The master pty is used by the networking application (z/OS UNIX) to communicate with the user applications, for example the shell. The slave pty is used by the shell process and associated applications. The TSO/E session reads and writes to the master, and the shell process reads and writes to the slave.

When a user uses the **OMVS** command to get into the z/OS UNIX shell, the user can use ISPF to edit (OEDIT) and browse (OBROWSE) HFS files. The user can switch between the shell and TSO session. Users can have multiple shell sessions at the same time and toggle between the sessions using PF keys.

The shell process can run in the same address space as the user's TSO/E session.

5.6 Invoking the shell via rlogin or telnet

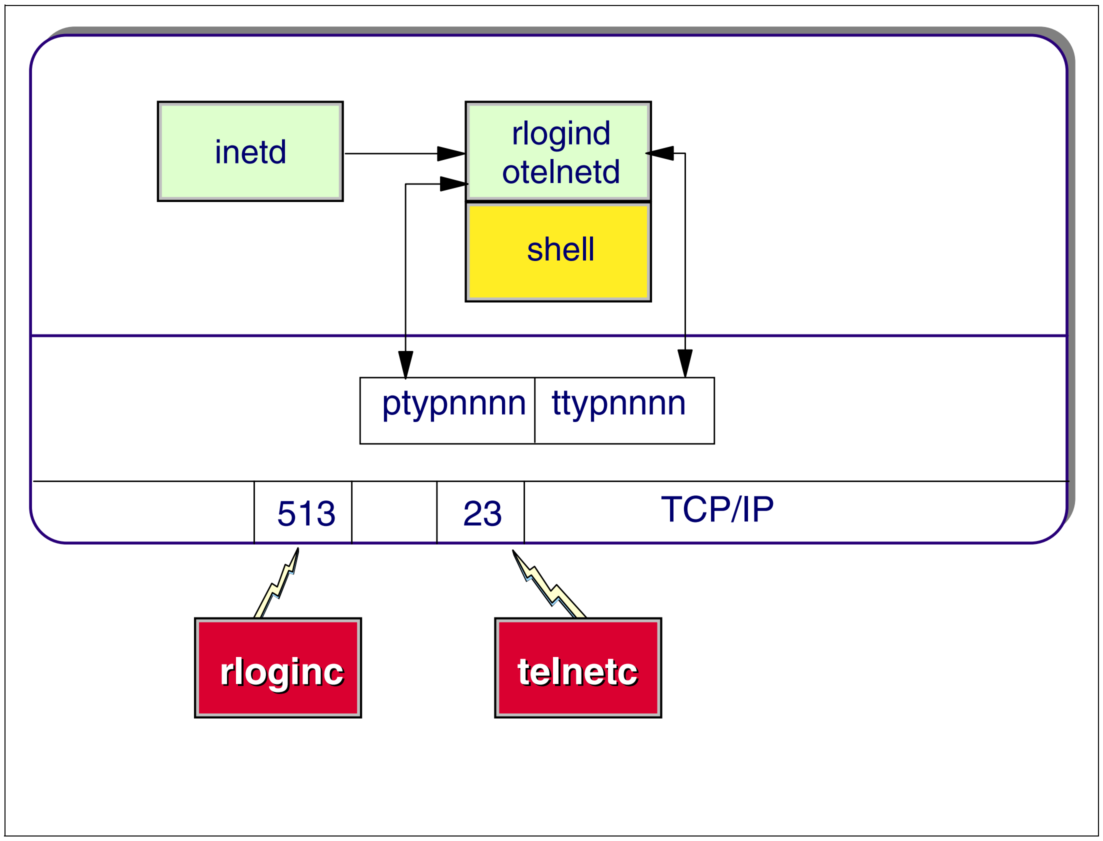


Figure 5-6 Invoking the shell from a rlogin or telnet

A z/OS UNIX user can log in directly to the z/OS UNIX shell in either of the following ways:

rlogin (remote login) If the inetd daemon is set up and active on the z/OS system, a workstation user with rlogin client support can use the TCP/IP network to log into the shell without going through TSO/E.

telnet Telnet support comes with the z/OS CS. It also uses the inetd daemon, which must be active and set up to recognize and receive the incoming telnet requests.

A z/OS system provides asynchronous terminal support for the z/OS UNIX shell. This is different from the 3270-terminal support provided by the TSO/E **OMVS** command. In a UNIX system, the UNIX shell operates in non-canonical mode (often called raw mode). This means that each keystroke is transmitted from the terminal to the system. Several UNIX applications, among them a popular editor called vi, depend on raw mode support. Using the 3270-terminal interface for the z/OS UNIX shell meant that UNIX applications depending upon raw mode support could not be ported to z/OS.

Figure 5-6 shows an overview of the two methods for logging in directly to the shell. Directly logging in to the shell is the only way to get raw mode support. Both rlogin and telnet require the inetd daemon to be set up and active.

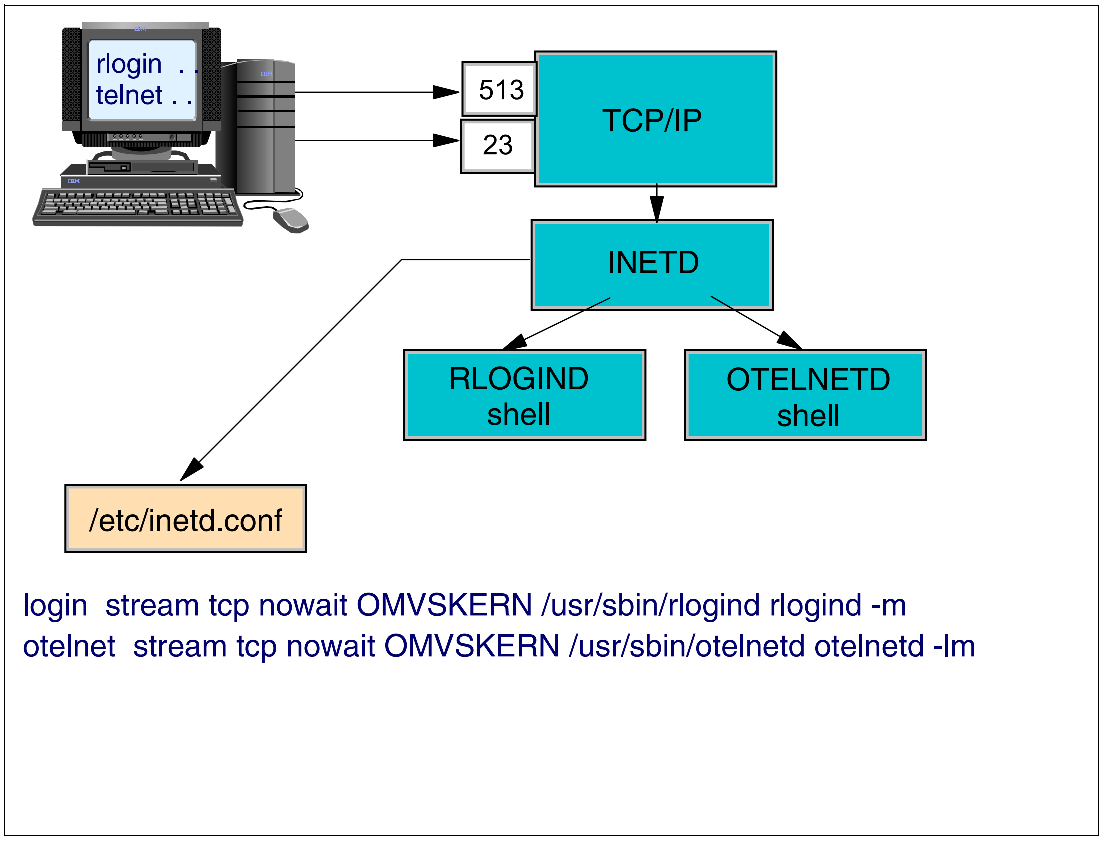
There are some differences between asynchronous terminal support (direct shell login) and 3270-terminal support (**OMVS** command):

- You cannot switch to TSO/E. However, you can use the TSO shell command to run a TSO/E command from your shell session.

- You cannot use the ISPF editor (this includes the oedit and TSO/E OEDIT commands, which invoke ISPF edit).

Note: The asynchronous interface does not automatically put you in raw mode. The terminal is set to operate in line mode, and only when using a utility that requires raw mode will it be changed.

5.7 rlogin and telnet access



```
login  stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
otelnet  stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -lm
```

Figure 5-7 rlogin and telnet access to the shell

The client user issues a **`telnet`** or **`rlogin`** command in the syntax of the system they are logged on to. TCP/IP must be set up on both systems to recognize the appropriate system names and ports.

The inetd daemon provides service management for a network. For example, it starts the rlogind program whenever there is a remote login request from a workstation.

The rlogind program is the server for the remote login command **`rlogin`** commonly found on UNIX systems. It validates the remote login request and verifies the password of the target user. It starts a z/OS shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

When inetd is running and receives a request for a connection, it processes that request for the program associated with that socket. For example, if a user tries to log in from a remote system into the z/OS shell while inetd is running, inetd processes the request for connection and then issues a fork() and execl() to the rlogin program to process the rlogin request. It then goes back to monitoring for further requests for those applications that can be found as defined in the /etc/inetd.conf file as follows:

```
login stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -lm
```

where:

stream stream or dgram. This is the socket_type.

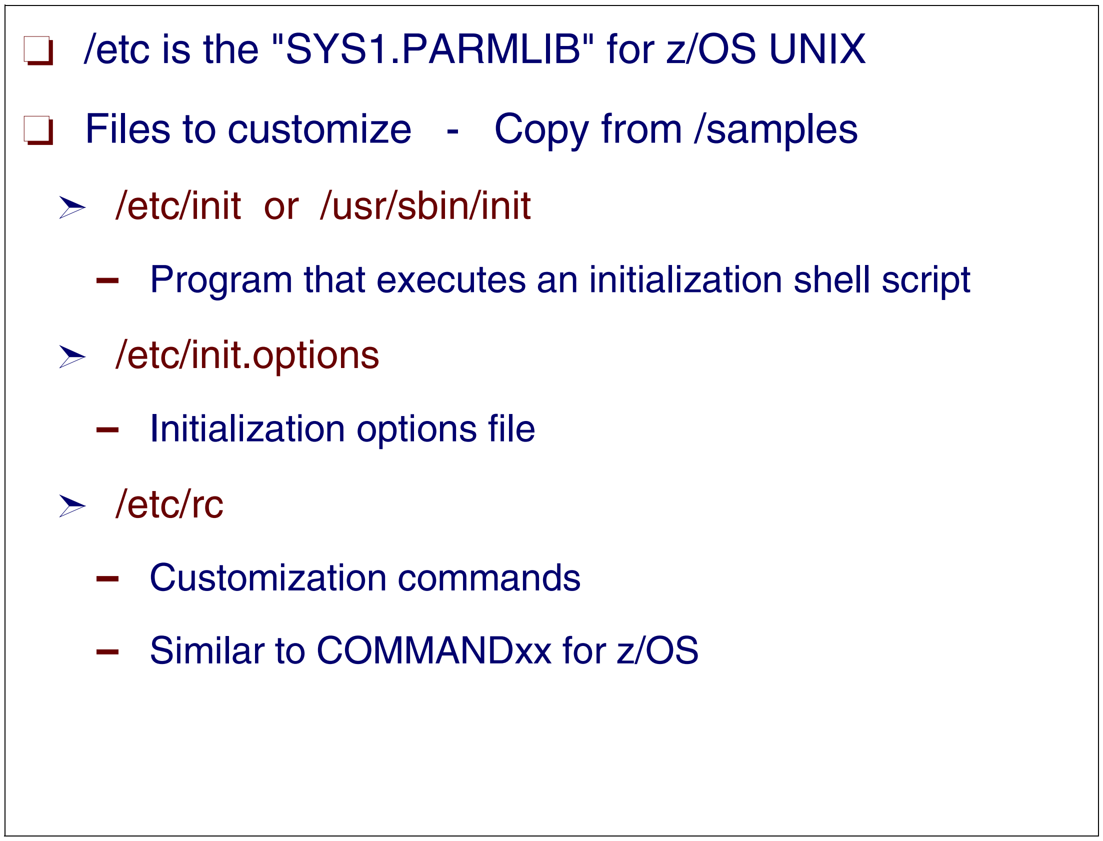
tcp	tcp or udp. The protocol is used to further qualify the service name. Both the service name and the protocol should match an entry in the data set, /etc/services, which is part of the user's TCP/IP configuration.
nowait	wait or nowait. Wait indicates the daemon is single-threaded and another request will not be serviced until the first one completes. Nowait specifies that the inetd daemon issues an accept when a connect request is received on a stream socket. If wait is specified, the inetd daemon does not issue the accept. It is the responsibility of the server to issue the accept if this is a stream socket.

The rlogin daemon or the telnet daemon is responsible for validating the user, handling ASCII-to-EBCDIC translation, and invoking the shell.

The presence of the **`-m`** option on the command indicates that the shell should start in the same address space as the daemon. The **`-l`** sets default mode for login to line mode.

A pseudo-TTY pair is assigned from the same pool to the shell session. To get multiple shell sessions the user issues another **`rlogin`** or **`telnet`** client command.

5.8 Customizing z/OS UNIX initialization



- ❑ /etc is the "SYS1.PARMLIB" for z/OS UNIX
- ❑ Files to customize - Copy from /samples
  - ➢ /etc/init or /usr/sbin/init
    - – Program that executes an initialization shell script
  - ➢ /etc/init.options
    - – Initialization options file
  - ➢ /etc/rc
    - – Customization commands
    - – Similar to COMMANDxx for z/OS

Figure 5-8 Files needed for z/OS UNIX initialization

Before continuing with customization, copy the file /samples/init.options to /etc/init.options. At startup, the kernel services attempt to run the program /etc/init, which is an initialization program created by the user. If that program is not found, then the kernel services try to run /usr/sbin/init, the default initialization program. Throughout this book, 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 for z/OS UNIX users. 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.

/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.

5.9 Initializing z/OS UNIX

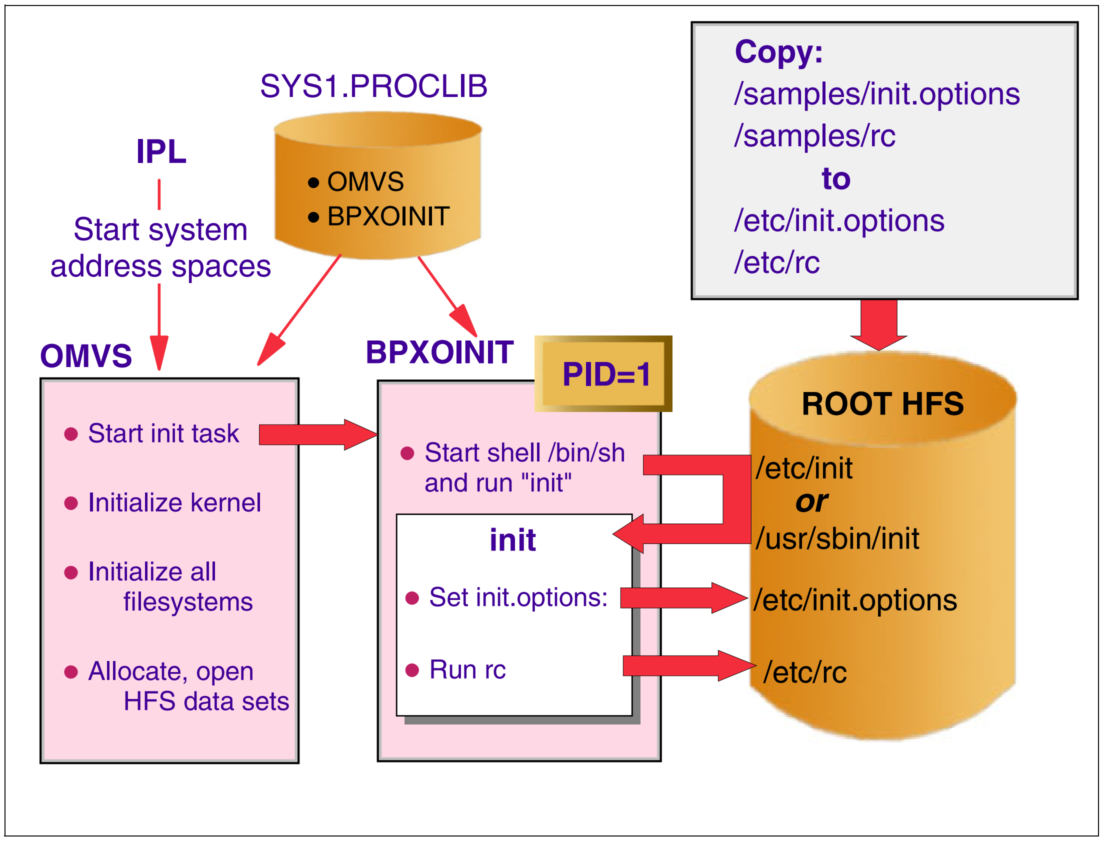


Figure 5-9 z/OS UNIX initialization processing

When z/OS UNIX is started, initialization includes running the /etc/init program. The file /etc/init is referred to as the initialization program that is run when the z/OS UNIX component is started, even though the /usr/sbin/init file may really be run.

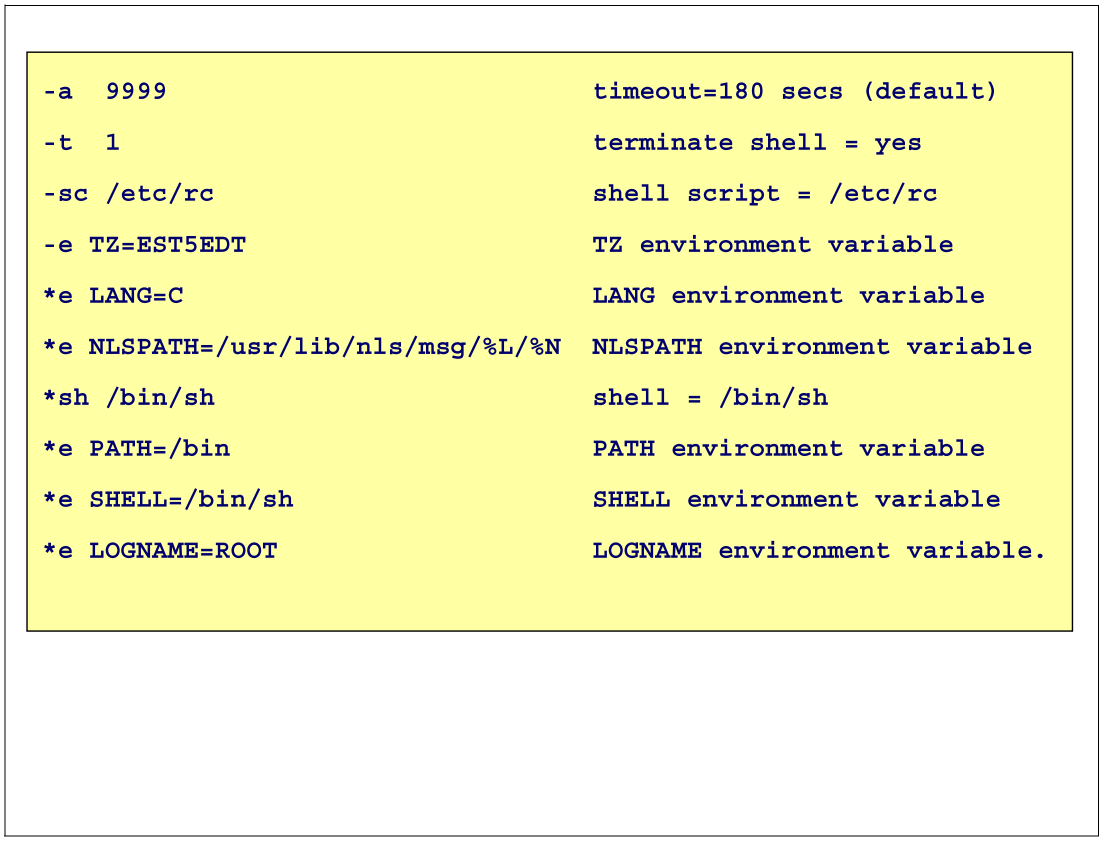
z/OS UNIX attempts to run the program /etc/init. If no such program is found, z/OS UNIX attempts to run /usr/sbin/init. This file contains the default initialization program shipped with the z/OS UNIX Shell and Utilities.

The /etc/init program invokes a shell to execute an initialization shell script that customizes the environment for shell users. When this shell script finishes or a time interval established by /etc/init expires, z/OS UNIX becomes available for general batch and interactive use.

Beginning with OS/390 V2R3, an option to use a REXX exec in an MVS data set was provided as an alternative to writing a customized /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.

The /usr/sbin/init program invokes a shell to execute an initialization shell script that customizes the environment for shell users. 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.

5.10 The /etc/init.options file



```
-a  9999                           timeout=180 secs (default)
-t  1                              terminate shell = yes
-sc /etc/rc                        shell script = /etc/rc
-e TZ=EST5EDT                      TZ environment variable
*e LANG=C                          LANG environment variable
*e NLSPATH=/usr/lib/nls/msg/%L/%N  NLSPATH environment variable
*sh /bin/sh                        shell = /bin/sh
*e PATH=/bin                       PATH environment variable
*e SHELL=/bin/sh                   SHELL environment variable
*e LOGNAME=ROOT                    LOGNAME environment variable.
```

Figure 5-10 The /etc/init.options file

The IBM-supplied default is located in the file /samples/init.options. Copy this file to /etc/init.options. This file will be used the next time z/OS UNIX is started.

The file /etc/init.options treats all lines in the file that do not start with a hyphen (-) as comment lines. Lines that start with a hyphen are used to specify options.

The following options can be specified:

-a The maximum time in seconds that /etc/init will wait for the initialization shell script to complete.

-t Terminate option indicating whether to terminate the script if the timeout occurs.

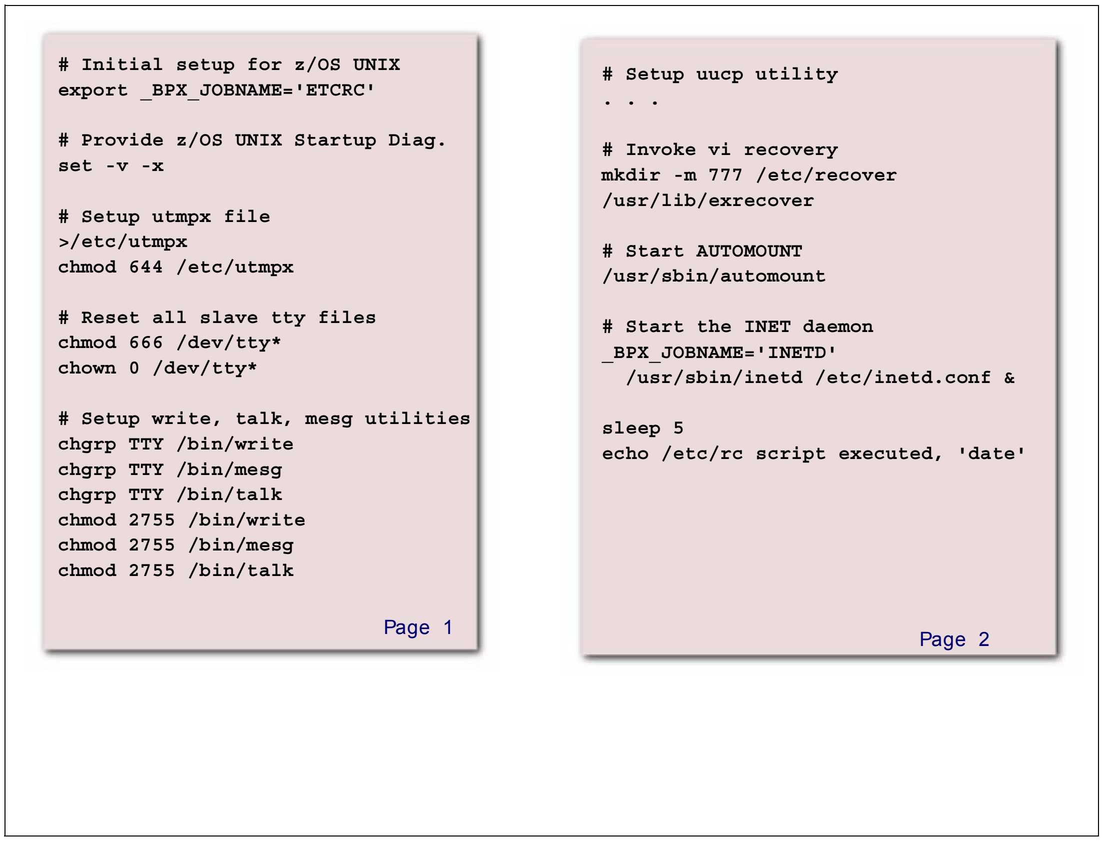
sh The pathname of the initializing shell.

-sc The pathname of the initializing shell script.

-e TZ=EST5EDT The time zone environment variable is set to the time for the USA east coast (five hours east of GMT/UTC). For a detailed description of how to set the time zone variable, see “Setting up the Time Zone Variable,” in *z/OS UNIX System Services Planning*, GA22-7800.

e Environment variable options. The environment variables that are set in this file will only be valid for the initialization shell. Later figures show how environment variables can be set for the shell users.

5.11 The etc/rc file



```
# Initial setup for z/OS UNIX
export _BPX_JOBNAME='ETCRC'

# Provide z/OS UNIX Startup Diag.
set -v -x

# Setup utmpx file
>/etc/utmpx
chmod 644 /etc/utmpx

# Reset all slave tty files
chmod 666 /dev/tty*
chown 0 /dev/tty*

# 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
```

Page 1

```
# Setup uucp utility
. . .

# Invoke vi recovery
mkdir -m 777 /etc/recover
/usr/lib/exrecover

# Start AUTOMOUNT
/usr/sbin/automount

# Start the INET daemon
_BPX_JOBNAME='INETD'
  /usr/sbin/inetd /etc/inetd.conf &

sleep 5
echo /etc/rc script executed, 'date'
```

Page 2

Figure 5-11 The /etc/rc file used during initialization

A sample initialization shell script can be found in the file /samples/rc. Copy this file to the /etc directory. No changes will be used until the next time z/OS UNIX is started.

The initialization script can be used to start daemons, or perform shell commands or scripts automatically each time z/OS UNIX is started. For example, the inetd daemon can be added to this script, as well as the automount command if it is to be used.

The /etc/rc file contains customization commands for z/OS UNIX System Services Application Services. The figure shows the IBM-supplied default in /samples/rc. Copy this file to /etc/rc and make the appropriate changes. If you already have an /etc/rc file, then compare it to /samples/rc and retrofit any new updates.

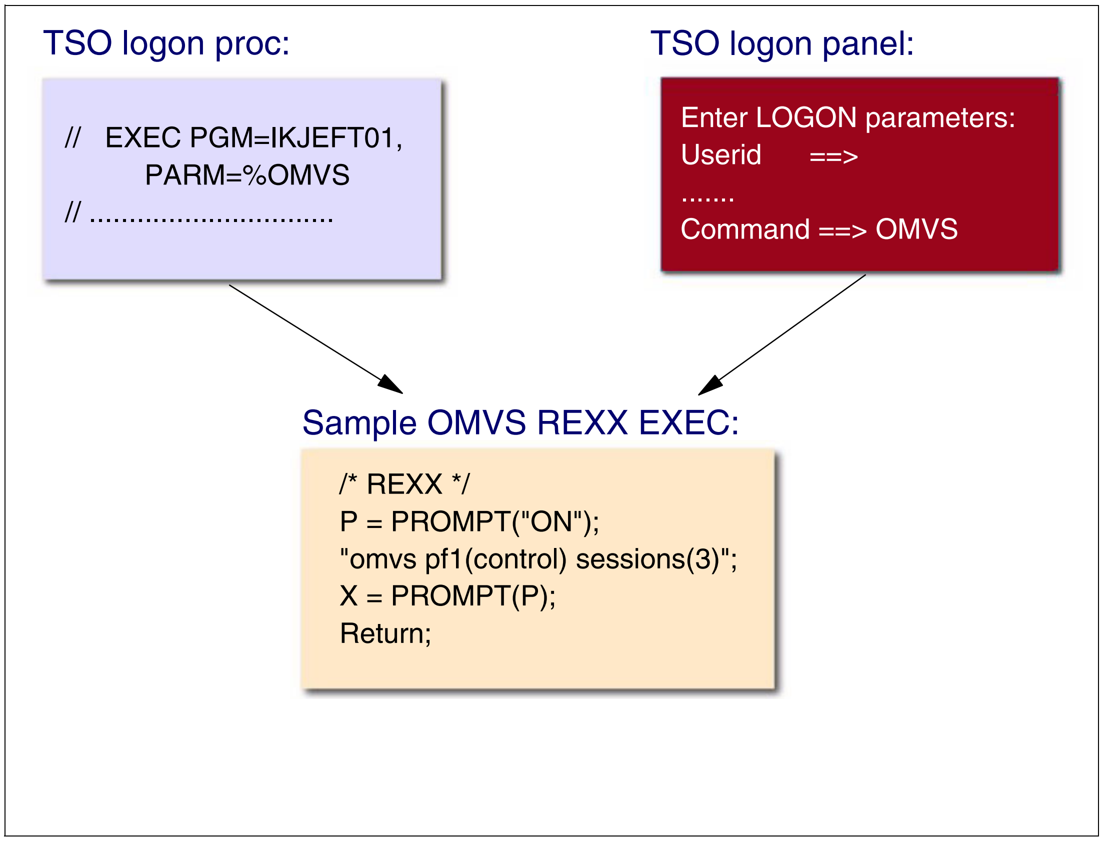
Customize your /etc/rc file by adding shell commands. For instance, you could add a command to start an installation-supplied daemon. The script can also invoke other scripts, for instance, an rc.tcpip script to start tcp daemons.

The _BPX_JOBNAME variable will set the z/OS job name for this script to ETCRC.

The visual shows parts of the sample provided in the /samples/rc file. The AUTOMOUNT facility is started and some lines were taken out to make it fit on the visual.

The example shows that the inetd daemon will be started automatically each time z/OS UNIX is initialized. The TCP/IP topic will have more information on the inetd start options. This _BPX_JOBNAME variable will set the z/OS job name to INETD.

5.12 Customizing the OMVS command



```
//   EXEC PGM=IKJEFT01,
        PARM=%OMVS
// ...............................
```

```
Enter LOGON parameters:
Userid      ==>
.......
Command ==> OMVS
```

```
/* REXX */
P = PROMPT("ON");
"omvs pf1(control) sessions(3)";
X = PROMPT(P);
Return;
```

Figure 5-12 Customizing the use of the **OMVS** *command from TSO/E*

The TSO/E access to the shell can be customized by the following options:

- The logon procedure for users that want to go directly into the shell when they log on to TSO/E can be modified to issue the **OMVS** command.
- A user can type the **OMVS** command on the TSO/E logon panel.
- The **OMVS** command has multiple options. A user can specify the **OMVS** command with the options he wants to use, or the system programmer can create a REXX EXEC called OMVS which will use a set of options that most users in an installation would prefer. For example:
  - Use of the 3270 alarm
  - Number of sessions (default is 1)
  - 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

When your profile allows for prompting, the PROMPT function can set the prompting option on or off for interactive TSO/E commands, or it can return the type of prompting previously set. When prompting is on, execs can issue TSO/E commands that prompt the user for missing operands.

The PROMPT function can be used only in REXX execs that run in the TSO/E address space. To set the prompting option on, use the PROMPT function followed by the word `'ON'` enclosed within parentheses.

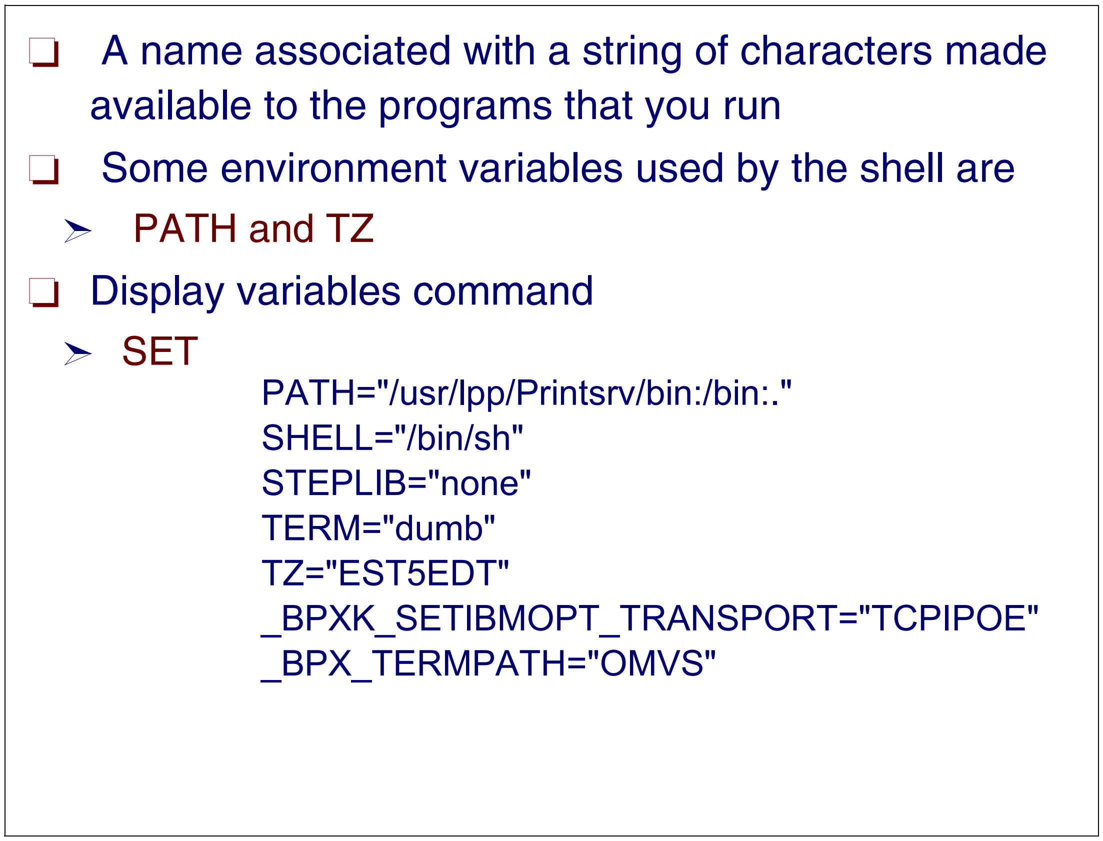
```
P = PROMPT('ON')
```

To reset prompting to a specific setting saved in variable `P`, write:

```
X = PROMPT(P)
```

Note: The time limit for a shell user is the same as the TSO/E timeout.

5.13 Environment variables



- A name associated with a string of characters made available to the programs that you run
- Some environment variables used by the shell are
  - PATH and TZ
- Display variables command
  - SET

```
PATH="/usr/lpp/Printsrv/bin:/bin:."
SHELL="/bin/sh"
STEPLIB="none"
TERM="dumb"
TZ="EST5EDT"
_BPXK_SETIBMOPT_TRANSPORT="TCPIPOE"
_BPX_TERMPATH="OMVS"
```

Figure 5-13 Environment variables

When a program begins, an environment is made available to it. The environment consists of strings of the form `name=value`, where `name` is the name associated with the environment variable, and its `value` is represented by the characters in value. UNIX systems traditionally pass information to programs through the environment variable mechanism.

There are global variables for all shell users and each user can override these variables with an individual set of variables. You can also change any of the values for the duration of your session (or until you change them again). You enter the name of the environment variable and equate it to a new value.

5.14 Code page tables

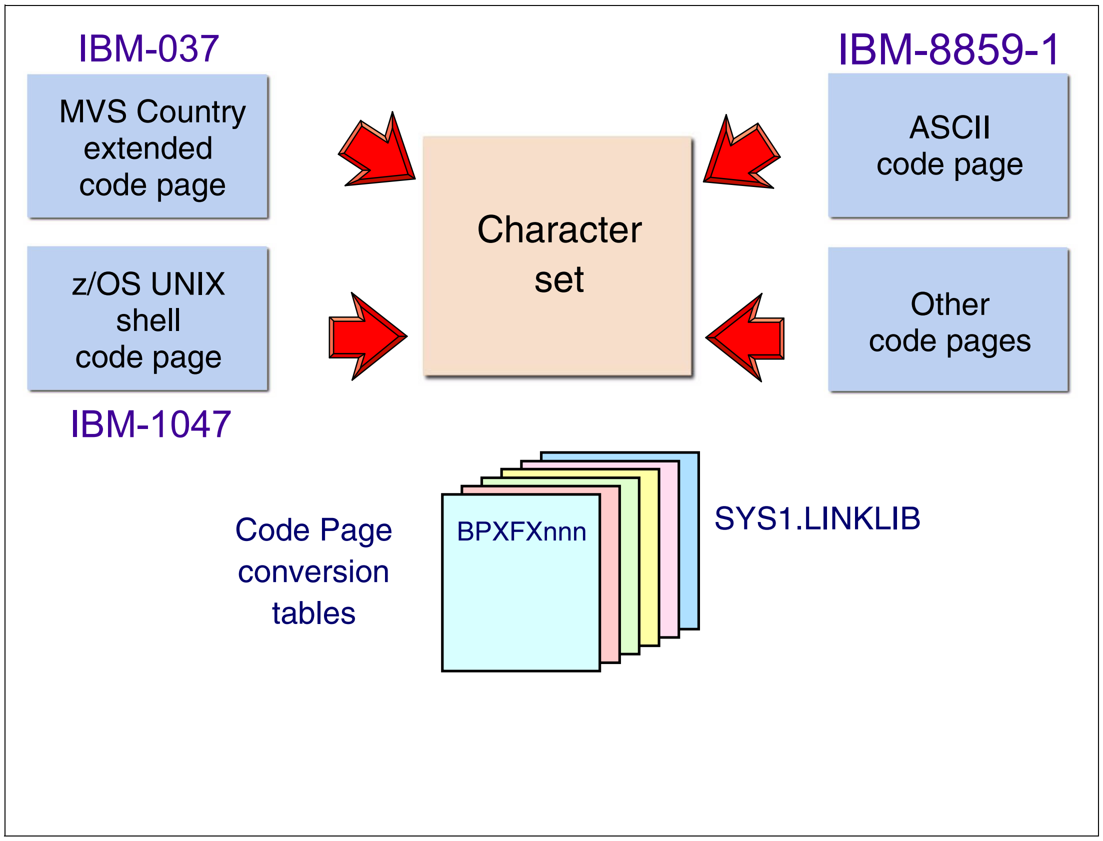


Figure 5-14 Defining code page tables

A code page for a character set determines the graphic characters produced for each hexadecimal code. The code page is determined by the programs and national languages being used.

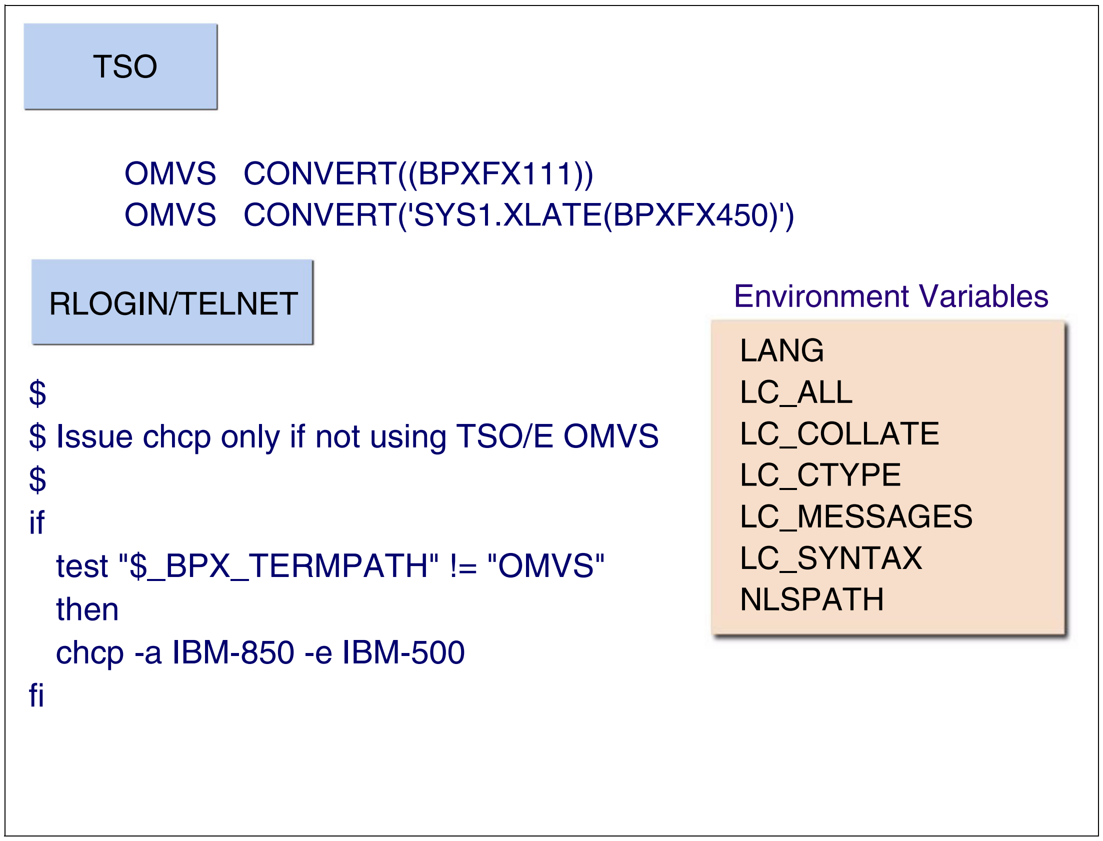
If the shell is using a locale generated with code pages IBM-1047, an application programmer needs to be concerned about “variant” characters in the POSIX portable character set whose encoding may vary from other EBCDIC code pages. For example, the encodings for the square brackets do not match on code pages IBM-037 and IBM-1047.

For most countries, MVS uses one code page and the z/OS UNIX shell uses a different one. To complicate the matter, many users have workstations which use an ASCII-based code page. When moving data between these environments, the data may have to be converted between the code pages to maintain the same characters. This may result in different hexadecimal codes depending on the code pages.

There are code page conversion tables located in SYS1.LINKLIB which can be used to convert between the z/OS code page and the shell code page for the national languages supported by the z/OS UNIX shell.

The source for these code page tables can be found in SYS1.SAMPLIB. They can be used as samples for creating a new table if an installation has special requirements.

5.15 Specifying a code page



```
OMVS   CONVERT((BPXFX111))
OMVS   CONVERT('SYS1.XLATE(BPXFX450)')
```

```
$
$ Issue chcp only if not using TSO/E OMVS
$
if
  test "$_BPX_TERMPATH" != "OMVS"
  then
  chcp -a IBM-850 -e IBM-500
fi
```

```
LANG
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_SYNTAX
NLSPATH
```

Figure 5-15 Specifying code pages

The **OMVS** command has a CONVERT option that lets you specify a conversion table for converting between code pages for the shell accesses by a TSO/E user. The table you want to specify depends on the code pages you are using in MVS and in the shell. For example, if you are using code page IBM-037 in MVS and code page IBM-1047 in the shell, specify the following when you enter the **OMVS** command:

```
OMVS CONVERT((BPXFX111))
```

The BPXFX111 translation table is for z/OS (code page 00033) to z/OS UNIX (code page 1047) translation. This would be used by most shell users in the U.S.

IBM has supplied many code pages for different countries with the product. If you were in Switzerland you might invoke the shell with the BPXFX450 conversion table. If you leave out the data set name, the normal z/OS search order is used to find the module.

If you are accessing the shell through the rlogin or telnet interface you do not use the **OMVS** command. You must change to the appropriate code page by issuing the **chcp** command. This could be issued by the user or be put in a profile for them.

The example in Figure 5-15 shows the code for a script that checks to be certain that **OMVS** was not issued, and then issues the **chcp** with an ASCII conversion table of IBM-850 and an EBCDIC table of IBM-500. You may also want to change the environment variables that reflect local formats at the same time.

5.16 Shell environment variables

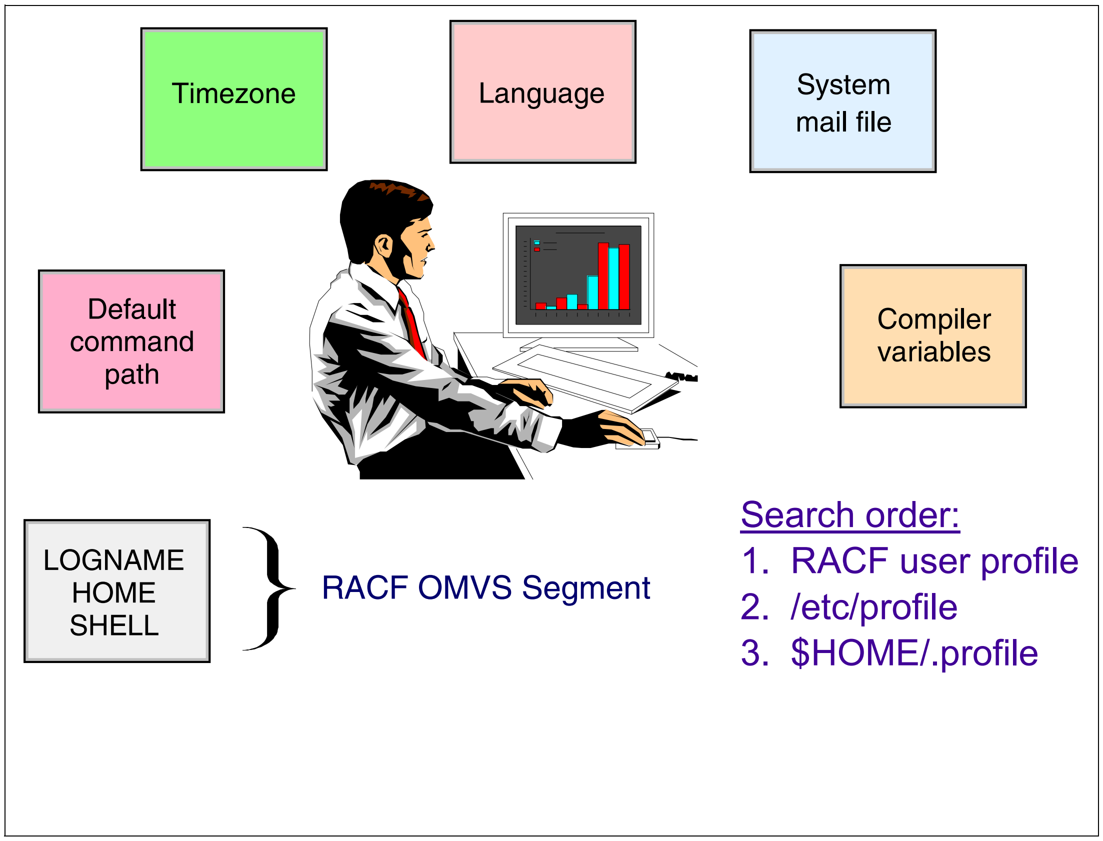


Figure 5-16 Specifying environment variables for the shell user

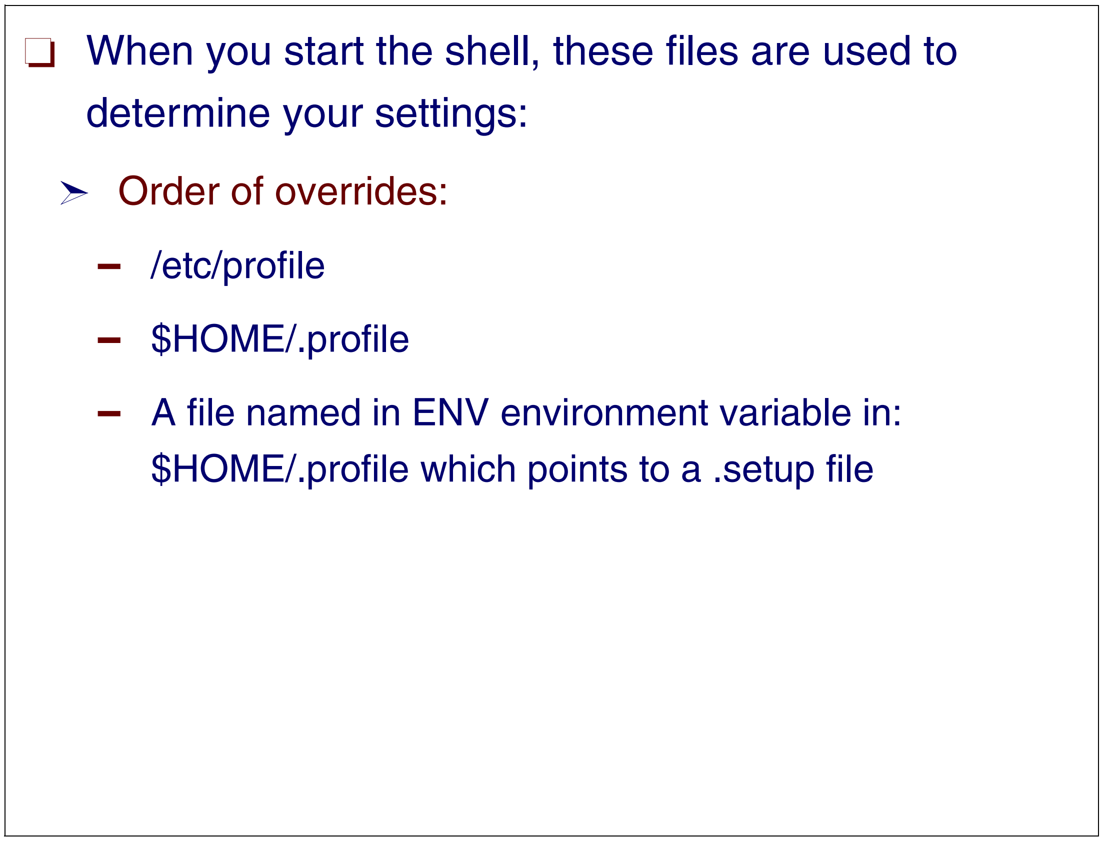
An environment variable is a variable that describes the operating environment of a process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone.

Depending on the commands used to set it, an environment variable can be global (used for all processes), local (used only for the current process), or can be exported (used for the current process and for any other processes created by the current process).

Setting an environment variable is optional. If a variable is not set, it will have no value. The following is a list of the places where environment variables can be set:

- By the system programmer:
  - RACF user profile
  - /etc/profile
- By the shell users:
  - $HOME/.profile
  - The file named in the ENV environment variable in $HOME/.profile
  - A shell command or shell script

5.17 Customizing your shell environment



- When you start the shell, these files are used to determine your settings:
  - Order of overrides:
    - /etc/profile
    - $HOME/.profile
    - A file named in ENV environment variable in: $HOME/.profile which points to a .setup file

Figure 5-17 Customizing the user shell environment

When you start the z/OS shell, it uses information in three files to determine your particular needs or preferences as a user. The files are accessed in this order:

1. /etc/profile
2. $HOME/.profile
3. The file that the ENV variable specifies

Settings established in a file accessed earlier can be overwritten by the settings in a file accessed later.

The /etc/profile file provides a default systemwide user environment. The system programmer may modify the variables in this file to reflect local needs (for example, the time zone or the language). If you do not have an individual user profile, the values in the /etc/profile are used during your shell session.

The $HOME/.profile file (where $HOME is a variable for the home directory for your individual user ID) is an individual user profile. Any values in the .profile file in your home directory that differ with those in the /etc/profile file override them during your shell session. z/OS provides a sample individual user profile. Your administrator can set up such a file for you, or you can create your own.

5.18 Global variables in /etc/profile

```
# Improve shell performance
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
# Set the time zone as appropriate.
TZ=EST5EDT
# Set a default command path, including current working dir (CDW)
PATH=/bin:.
# Sets the path environment variables
LIBPATH=/lib:/usr/lib:.
MANPATH=/usr/man/%L
NLSPATH=/usr/lib/nls/msg/%L/%N
# Sets the language
LANG=C
# Sets the name of the system mail file and enables mail notification.
MAIL=/usr/mail/$LOGNAME
# Export the values so the system will have access to them.
export TZ PATH LIBPATH MANPATH NLSPATH MAIL LANG
# Set the default file creation mask - umask
umask 022
# Set the LOGNAME variable readonly so it is not accidentally modified.
readonly LOGNAME
```

Figure 5-18 Customizing the global /etc/profile variables for all users

All shell users will get the environment variables set by this profile. Users can override some of the variables by setting the variables in their private profiles or in their shell sessions.

Note: All environment variables are written with capital letters.

The /etc/profile file contains the environment variables and commands used by most shell users. An IBM-supplied sample is located in /samples/profile. Copy the sample to the /etc directory and make any necessary changes.

STEPLIB The use of STEPLIB=none is recommended to improve performance for shell users. Keep this setting.

TZ The timezone (TZ) is based on the Greenwich Mean Time (GMT), also called Universal Time Coordinated (UTC). Change it to your local time. The time zone used in the example is for the US East Coast (five hours west of UTC or GMT). It also shows the daylight saving time zone that will be used (EDT).

PATH This specifies the default command path where z/OS UNIX will search for the commands and programs a user is executing. If your installation will have programs in for example, /usr/bin, specify:

```
PATH=/bin:/usr/bin
```

In the PATH variable, a colon (:) separates the list of pathnames. The dot (.) represents the current working directory for a user. The order listed in the PATH variable controls the search order.

NLSPATH	Specifies the pathname for message catalogs.
LANG	Specifies the name of the default locale.
MAIL	Specifies the pathname for mail files. Change it if you want to use another pathname for the mail files. The setting for the MAIL variable indicates that each user will use a mail file with the same name as their LOGNAME, and it will be located in the directory specified for the variable. Note that when you use a dollar sign ($) in front of a variable in a shell script, it means that you want to use the value assigned to that variable. $LOGNAME will be interpreted as the user ID of the shell user (see LOGNAME defined in RACF profile).
EXPORT	The **`export`** command will export a variable to a subshell. All the environment variables defined will be valid only for the shell session (login shell) a user gets when the user invokes the shell (TSO **`OMVS`** or **`rlogin`**). Shell scripts and many shell commands will actually be executed in a separate shell created by the login shell. The shell creates child processes to run the scripts or commands. These subshells will not be aware of the variable settings that were specified in the login shell unless you export the variable specifications. To ensure that the subshell runs with the same environment, use the **`export`** command to export the variables.
UMASK	The **`umask`** command was introduced in our discussion on security. The umask value of 022 will result in permission bits: rwxr-xr-x.

Note: Any changes made in /etc/profile will take affect the next time a user invokes the shell.

The contents of the /etc/profile are actually a shell script. It contains examples of the script programming language with if-then statements, it sets values for environment variables, and it issues some shell commands (**`export`**, **`umask`**, **`readonly`**).

5.19 Setting the time zone

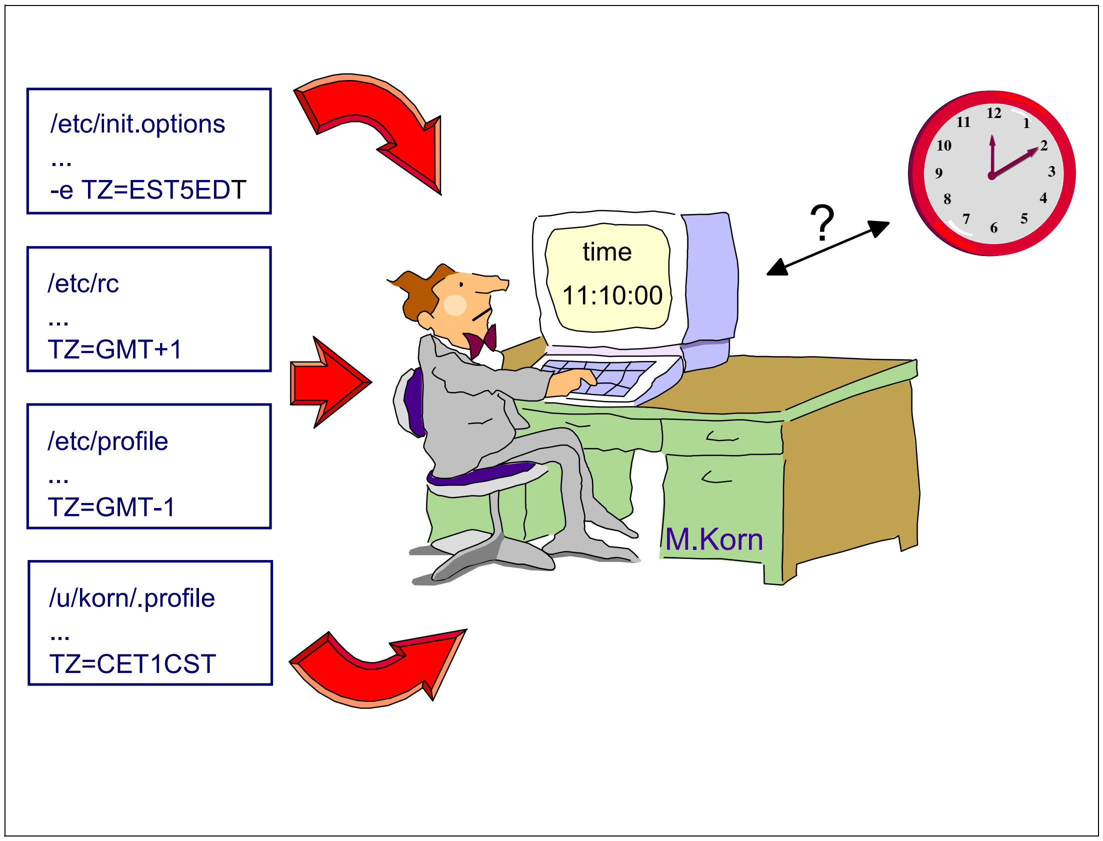


Figure 5-19 Setting the time zone variable

The time zone is an environment variable used by the time service to set the correct time for z/OS UNIX and for applications running on z/OS UNIX.

It is possible to specify a time zone variable in 4 different files. Table 5-1 shows the files and when changes made in these files become active.

Table 5-1 Activation schedule for time zone variables

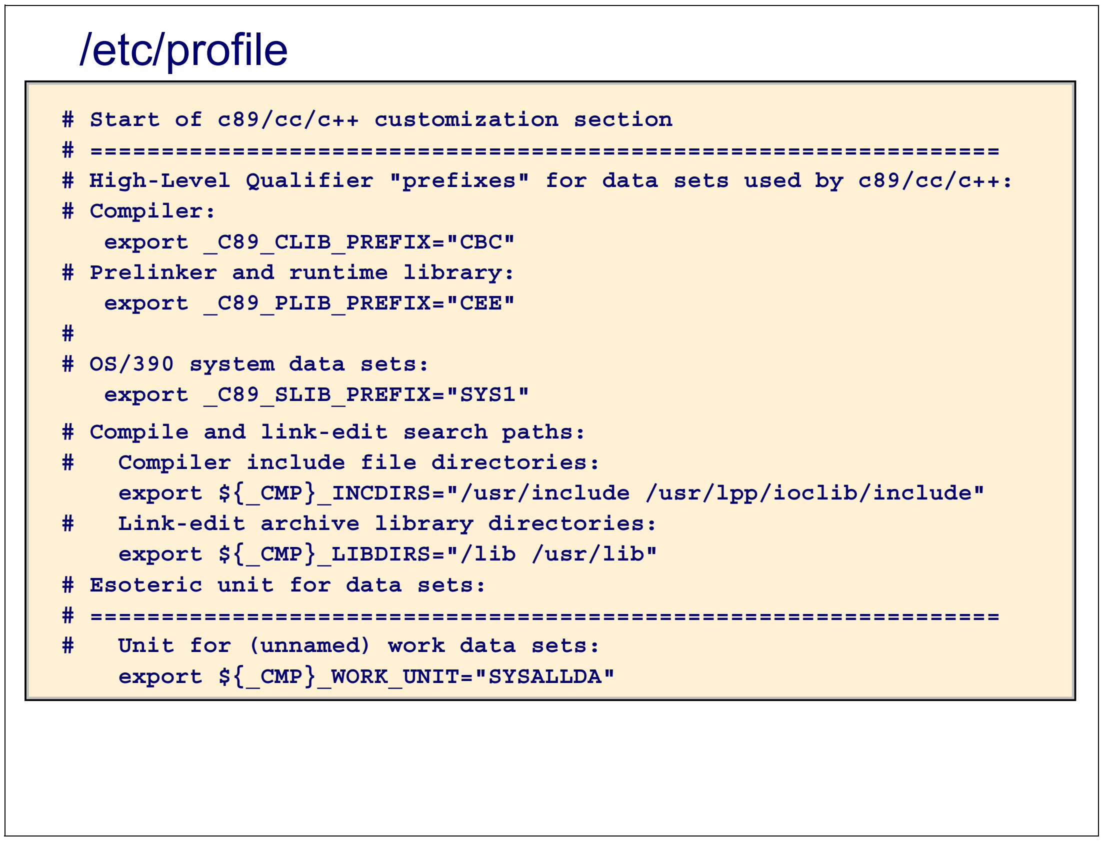
Files	**After z/OS UNIX restart**	**After next user enters the shell**	**Immediately after change**
/etc/init.options	YES	NO	NO
/etc/rc	YES	NO	NO
/etc/profile	YES	YES	NO
$HOME/.profile	YES	YES	NO

If the time zone variable is changed, you must restart z/OS UNIX or re-IPL z/OS.

General activation order: First the settings in /etc/init.options and /etc/rc become active at kernel initialization time. Afterwards, the settings in /etc/profile are read. If a user has a .profile in his HOME directory that has its own TZ value specified, this setting will be the active one for this user if he enters the shell.

Recommendation: It is recommended to have a TZ setting active in /etc/init.options and in /etc/profile. If there are daemons to start via the /etc/rc file, export the TZ variable before starting the daemons. If you want the correct time on any replies inside the shell, you will have to specify the TZ variable at least in the user profiles ($HOME/.profile) if no specifications were made in the system-wide profile /etc/profile where all users are affected.

5.20 Customizing the C89/CC compilers


/etc/profile

```
# Start of c89/cc/c++ customization section
# ================================================================
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# Compiler:
   export _C89_CLIB_PREFIX="CBC"
# Prelinker and runtime library:
   export _C89_PLIB_PREFIX="CEE"
#
# OS/390 system data sets:
   export _C89_SLIB_PREFIX="SYS1"
# Compile and link-edit search paths:
#   Compiler include file directories:
    export ${_CMP}_INCDIRS="/usr/include /usr/lpp/ioclib/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="SYSALLDA"
```

Figure 5-20 Customizing the c89/CC compiler options

As we mentioned earlier, the /etc/profile file provides a default system-wide user environment. Any environment variables that are set in /etc/profile affect every shell user unless they override these settings in their own $HOME/.profile or in a setup script file that is pointed to by the ENV variable.

For each z/OS release, there is a default compiler for c89, cc, or c++ (cxx) to use. Optionally, you can choose to use a non-default compiler. This section lists the compiler choices for each release, including the default compilers; the environment variable settings for each compiler are identified.

The c89/cc/c++ utilities use a number of environment variables. The default values are specified as comments in the /samples/csh.login file that is shipped with each release. The naming conventions for the environment variables are as follows:

- 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++ utility), then this will also target your executable for the same level of C/C++ Class Library.

In the section titled '`c89/cc/c++ customization section`', remove the comment for the following export commands to tell the C/C++ compiler what the high level qualifier of the C/C++, Language Environment and z/OS target library data sets are. Check the high level qualifiers on your system to make sure they agree with these default values.

Use ISPF 3.4 and look at your target libraries to check. If they agree, there is no need to uncomment these lines because they are the default setting.

```
#export _C89_CLIB_PREFIX="CBC"
#export _C89_PLIB_PREFIX="CEE"
#export _C89_SLIB_PREFIX="SYS1"
```

Note: SYS1 refers to the high level qualifier of CSSLIB, MACLIB and MODGEN on your system.

If your high level qualifiers do not match, then make the changes here and uncomment the three export statements. See the Environment Variables section of the c89/cc/c++ command discussion in *z/OS UNIX System Services Command Reference*, SA22-7802 for more information.

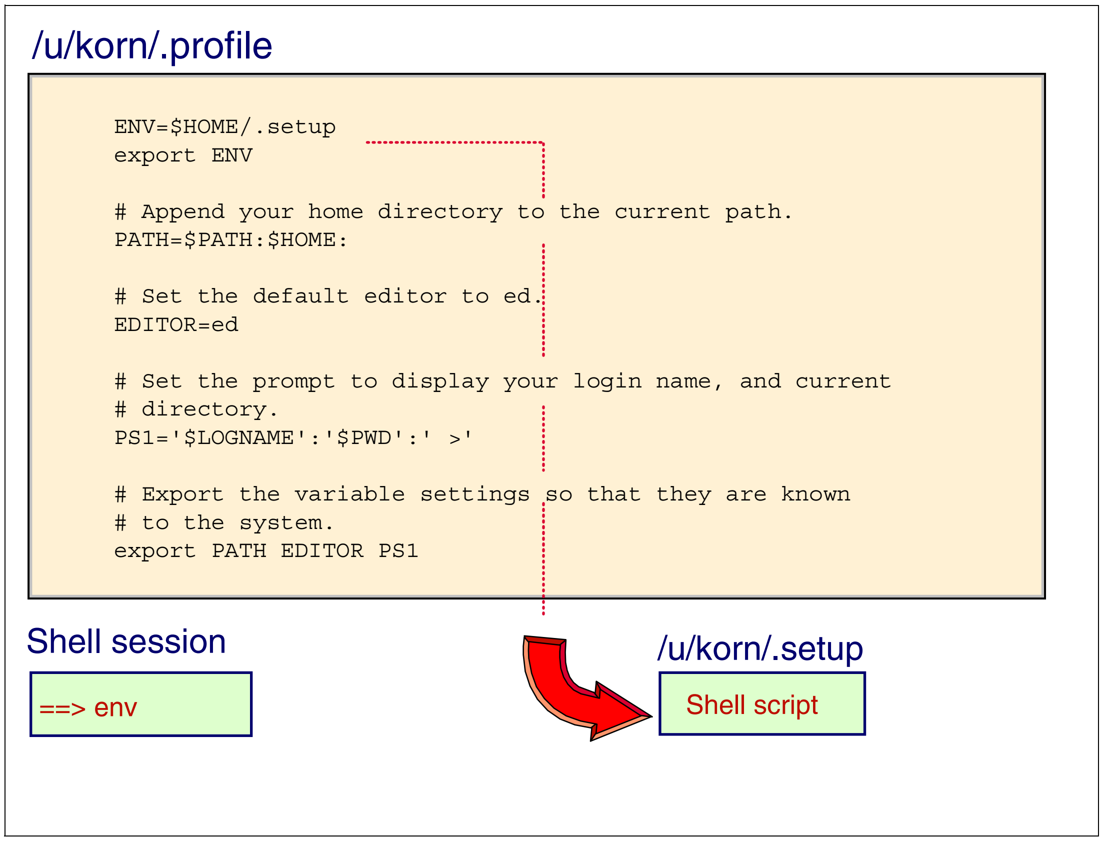
If the system is not configured with an esoteric unit SYSDA, or some other esoteric unit is to be used for VIO temporary unnamed work data sets set by c89, the following environment variable needs to be set. Specifying a null value for this variable ("") results in c89 using an installation-defined default for the UNIT. The environment variable is shown being set to the default value:

```
#export _C89_WORK_UNIT="SYSDA"
```

The environment variables used by the cc utility have the same names as the ones used by c89 except that the prefix is _CC instead of _C89. Likewise, for the c++ (cxx) utility, the prefix is _CXX instead of _C89. Normally, you do not need to explicitly assign the environment variables for all three utilities. These **eval** commands set the variables for the other utilities, based on those set for c89.

```
#eval "export $(typeset -x | grep "^_C89_" ...
#eval "export $(typeset -x | grep "^_C89_" ...
```

5.21 User-defined settings



```
ENV=$HOME/.setup
export ENV

# Append your home directory to the current path.
PATH=$PATH:$HOME:

# Set the default editor to ed.
EDITOR=ed

# Set the prompt to display your login name, and current
# directory.
PS1='$LOGNAME':'$PWD':' >'

# Export the variable settings so that they are known
# to the system.
export PATH EDITOR PS1
```

Figure 5-21 Setting the user variables by the user

Shell users can set their own values for the environment variables using any of the following methods:

- A $HOME/.profile file. A sample is provided in /samples/.profile/; it can be copied to a shell user's home directory and modified.

  Add the environment variable _BPX_SHAREAS=YES to the user profile. This will cause all shell commands that run in separate processes to create the processes in the same address space as the shell.

- The ENV variable in the $HOME/.profile can be set to a file name which is a shell script that sets environment variables.
- A shell user can use the **env** command to display the environment variables for their session, and to change any of these variables. The change will only last for the length of the session.

The .profile file must be located in a user's home directory. Figure 5-21 shows the contents of the sample provided by IBM. One of the variables in the .profile is called ENV, and this variable can be used to point to another file where a user can add environment variables and shell commands that the user wants to perform when the shell is invoked.

Some of the things a user may want to update in the .profile are:

- ENV: use if the user has a separate shell script with environment variables.
- PATH: Add the user's home directory to the search path:

```
PATH=$PATH:$HOME:
```

Note how variables can be used in other variables. System programmers and system administrators need to have a $HOME/.profile file with the PATH environment set as follows:

```
PATH=$PATH:/usr/sbin:$HOME:
```

This allows the system programmers to run authorized utilities and to start daemons found in /usr/sbin, as follows:

PS1	Change the shell prompt. The default is #.
TZ	Change if the user is located in a different time zone than the system.
_BPX_SHAREAS=YES	Add this to the profile to reduce the number of address spaces used by a shell user.

If using the **`env`** command to set a variable, the setting will only last for the duration of the user's session. The next time the user invokes the shell, this setting is forgotten. Use the .profile file or a shell script for variables that should be the same for each session.

The system programmer should use the /etc/profile file for variables that all users should have set.

5.22 Setting the region size

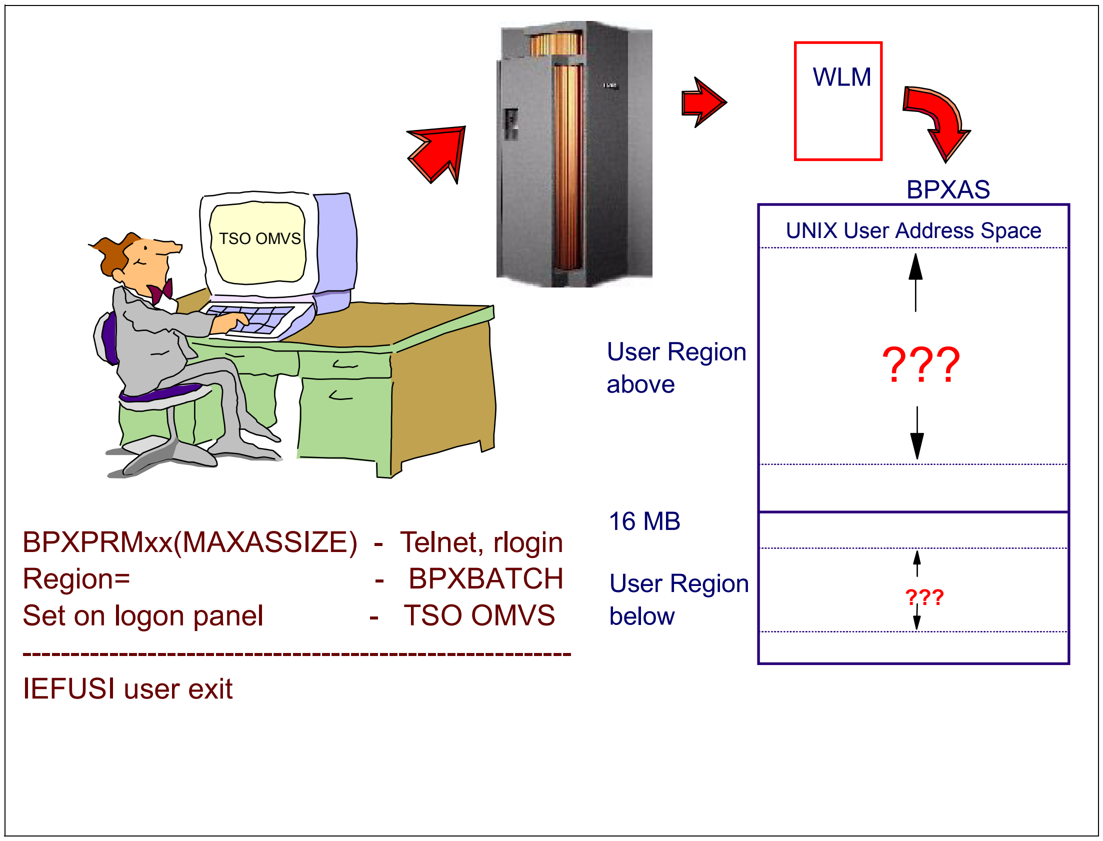


Figure 5-22 Setting the user region size default

The region size describes the amount of storage within which a user is allowed to run/execute programs. This value determines what kinds of programs (depending on their size) and how many programs are executable at the same time.

In principle, TSO users get the region size from the logon panel to their programs. If a user enters the z/OS UNIX shell by issuing a TSO `OMVS` command, the parent process doesn't run in the user's address space. Under control of the Work Load Manager, an address space will be created that gets the same region size as the TSO/E user itself.

If BPXBATCH is used there is a BPXBATCH REGION parameter that describes the region size; however, started procedures that create z/OS UNIX processes use the REGION parameter on the EXEC statement.

Telnet and rlogin users get their region size from the MAXASSIZE parameter in the SYS1.PARMLIB (BPXPRMxx) member.

IEFUSI is a user exit where an installation can set the region size and region limit for all programs that run under this job step. Make sure this exit does not change the region size setting for the z/OS UNIX process.

5.23 Setting up printers for shell users

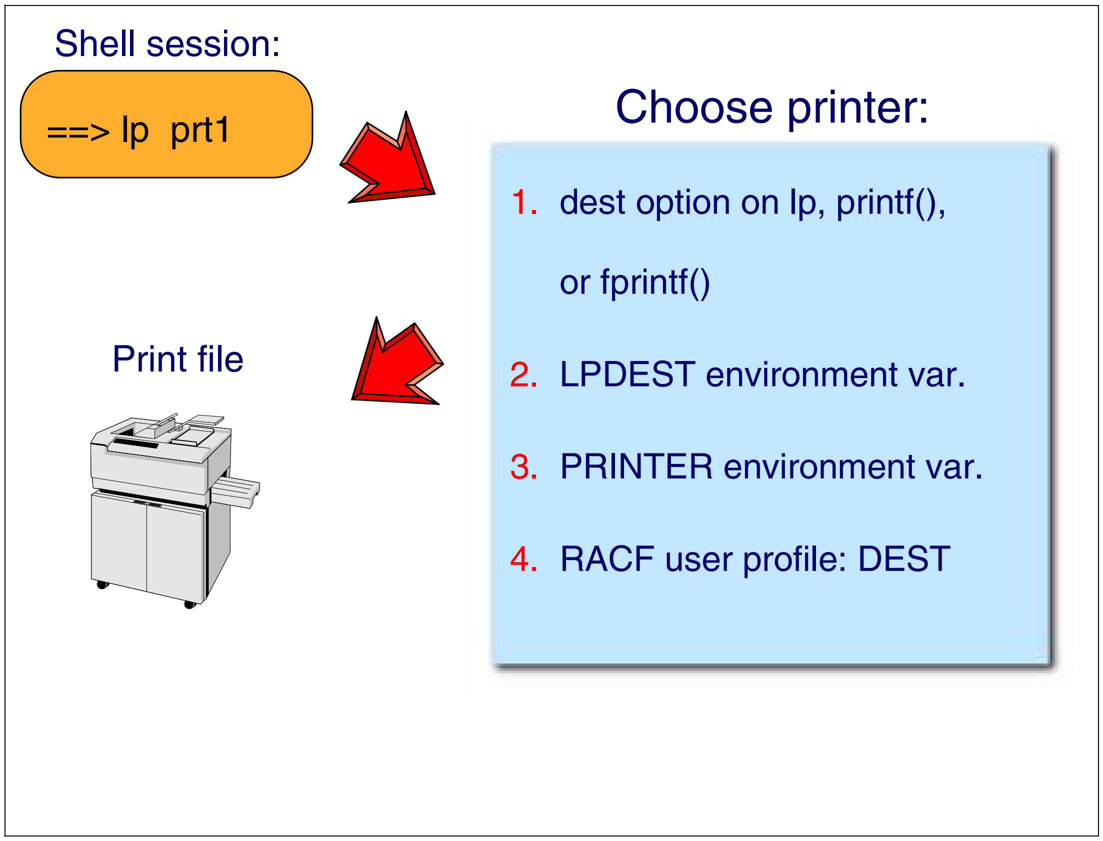

Shell session:
==> lp prt1
Print file
Choose printer:
1. dest option on lp, printf(), or fprintf()
2. LPDEST environment var.
3. PRINTER environment var.
4. RACF user profile: DEST

Figure 5-23 Shell user printer defaults

Each z/OS UNIX user has a number of default printers specified in different ways. The system will select a printer in the following order:

- The printer in the dest option of the `lp` shell command, or the printf() or fprintf() functions.
- The printer specified in the LPDEST environment variable.
- The printer specified in the PRINTER environment variable.
- The printer in the RACF user profile. It is specified by the DEST parameter of the RACF **ADDUSER** or **ALTUSER** command.

The z/OS Infoprint Server, available beginning with OS/390 V2R8, provides much greater support for printing than was available in previous systems.

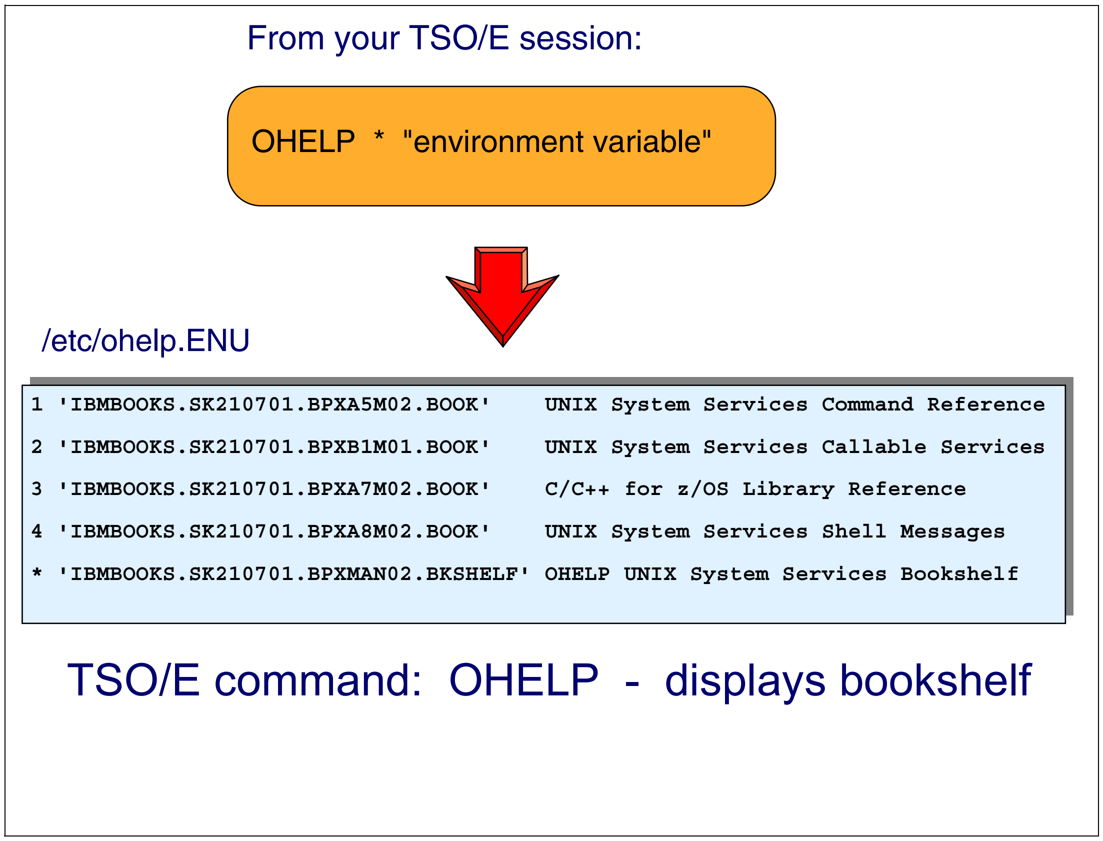
Inform the application programmers of the destinations or symbolic names of printers you specified in JES initialization statements. The dest option of the `lp` command uses the same destinations as the DEST parameter in the OUTPUT JCL statement. The dest option on `lp` can be:

- LOCAL for any installation printer.
- A destination that is defined in a JES2 DESTID initialization statement.

If omitted, the system uses the default printer.

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.

5.24 Installing books for OHELP


From your TSO/E session:
OHELP * "environment variable"
/etc/ohelp.ENU
TSO/E command: OHELP - displays bookshelf

```
1 'IBMBOOKS.SK210701.BPXA5M02.BOOK'    UNIX System Services Command Reference
2 'IBMBOOKS.SK210701.BPXB1M01.BOOK'    UNIX System Services Callable Services
3 'IBMBOOKS.SK210701.BPXA7M02.BOOK'    C/C++ for z/OS Library Reference
4 'IBMBOOKS.SK210701.BPXA8M02.BOOK'    UNIX System Services Shell Messages
* 'IBMBOOKS.SK210701.BPXMAN02.BKSHELF' OHELP UNIX System Services Bookshelf
```

Figure 5-24 Setting the user OHELP command to access the z/OS UNIX books

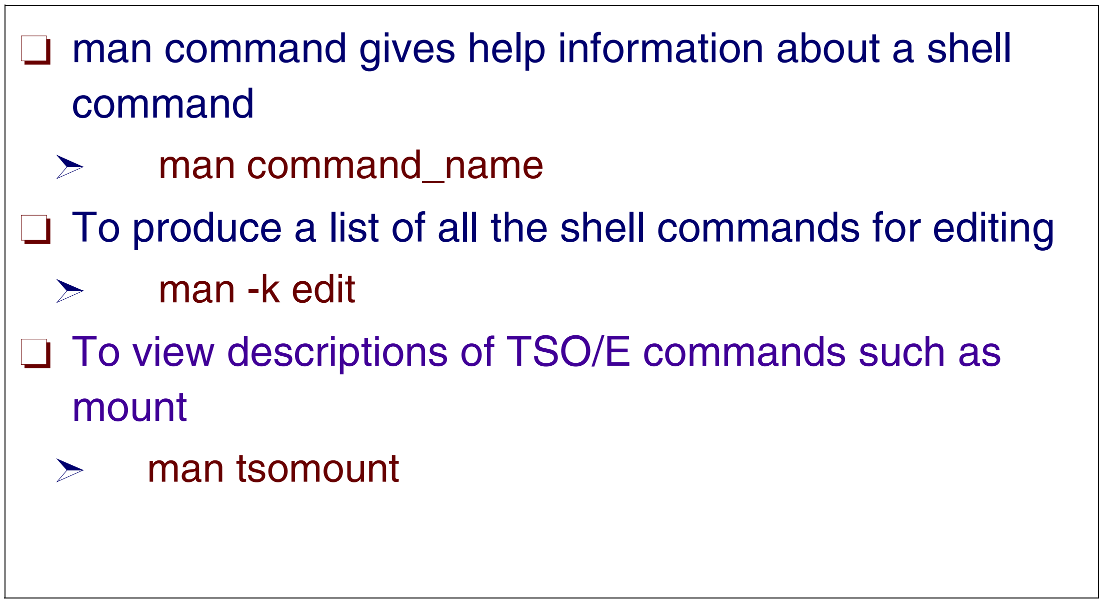
The TSO/E **OHELP** command provides online help for:

- Shell commands
- Callable services
- C functions
- Shell messages

In order to use **OHELP** the following must be done:

- BookManager READ/MVS is required.
- The z/OS UNIX bookshelf must be installed.
- An HFS file must be customized to map the OHELP reference ID with a z/OS UNIX book. Do this using the following steps:
  - Copy the sample file in /samples/ohelp.ENU to /etc/ohelp.ENU.
  - Update the file if you need to change the data set names of the online books, or if you want to add or delete books.

5.25 Using the man command



- man command gives help information about a shell command
  - man command_name
- To produce a list of all the shell commands for editing
  - man -k edit
- To view descriptions of TSO/E commands such as mount
  - man tsomount

Figure 5-25 Using the man command to access manuals

You can use the **man** command to get help information about a shell command. The man syntax is:

```
man command_name
```

To scroll the information in a man page, press Enter.

To end the display of a man page, type `q` and press Enter.

To search for a particular string in a system that has a list of one-line command descriptions, use the `-k` option:

```
man -k string
```

For example, to produce a list of all the shell commands for editing, you could type:

```
man -k edit
```

You can use the man command to view manual descriptions of TSO/E commands. To do this, you must prefix all commands with TSO. For example, to view a description of the **MOUNT** command, you would enter:

```
man tsomount
```

-k	Searches a precomputed database of syntax lines for information on keywords.
-M	`-M` *path* searches the directories indicated by path for help files. If **-M** is not specified, man uses the path specified in the MANPATH environment variable if set; otherwise man searches /usr/man/%L. The value of the LANG environment variable is substituted for %L in this directory and in the directories specified by MANPATH.
-w	Displays only the filename of the file containing the help file.
-x	Displays what files man is searching to find the help file. **-x** also displays any errors man encounters in extracting man pages from online book files.

5.26 Enabling various tools

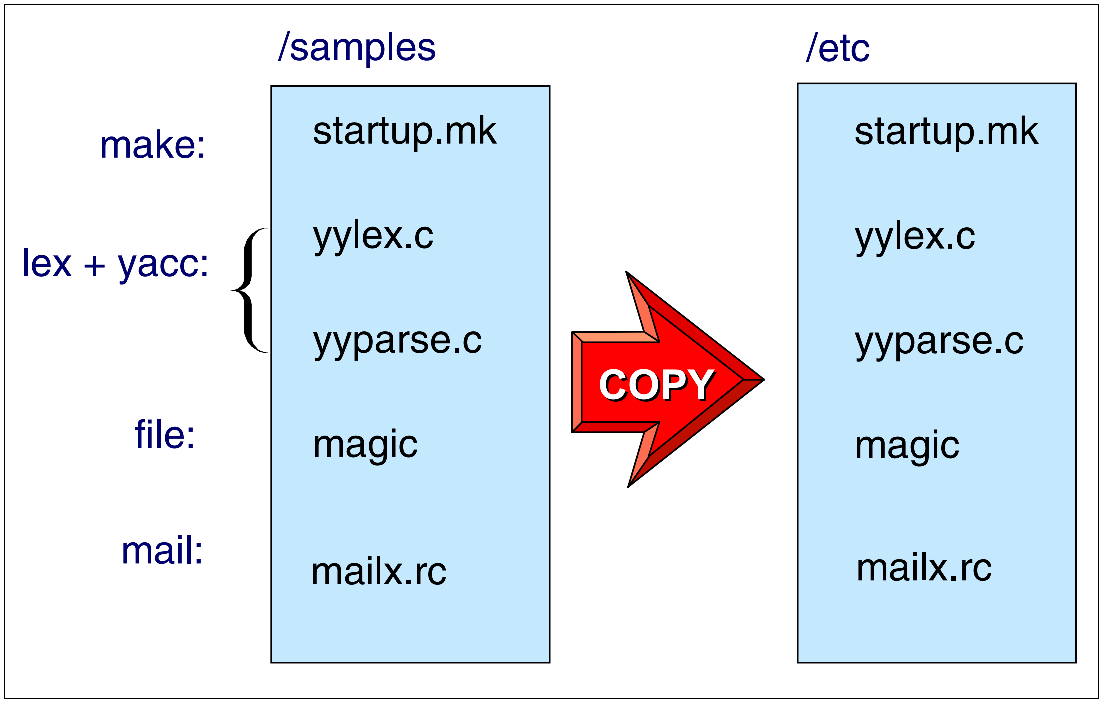


Figure 5-26 Customizing shell utilities

Some of the shell utilities must be customized before they can be used:

make The make utility uses a configuration file. Copy the sample in /samples/startup.mk to /etc/startup.mk.

The make utility helps manage large applications consisting of multiple programs. When an application programmer updates a program, make will help keep all the files/programs in synchronization.

lex For the lex utility, copy the sample in /samples/yylex.c to /etc/yylex.c. This file contains the prototype lex scanner.

lex and yacc are utilities that help an application programmer to write programs. For example, input to lex and yacc describes how you want the final program to work. The output from lex and yacc is C source code. lex and yacc are often used to develop programs that analyze and interpret input, for example a compiler.

yacc For the yacc utility, copy the sample in /samples/yyparse.c to /etc/yyparse.c. This file contains the C parser template used by yacc.

file For the file utility, copy the sample in /samples/magic to /etc/magic.c. This file contains a series of templates showing different file types.

The file utility determines the format of a file by inspecting the attributes and (for an ordinary file) reading an initial part of the file. It compares each file on the command line to templates found in a system-maintained magic file to determine their file type.

mailx For the mailx utility, copy the sample in /samples/mailx.rc to /etc/mailx.rc. This file contains variable values and definitions common to all shell users.

The mailx utility program allows users to send messages to one another. The mailbox name is set up in the user's profile. It defaults to /usr/mail/$LOGNAME.

5.27 IVP for z/OS UNIX and tools

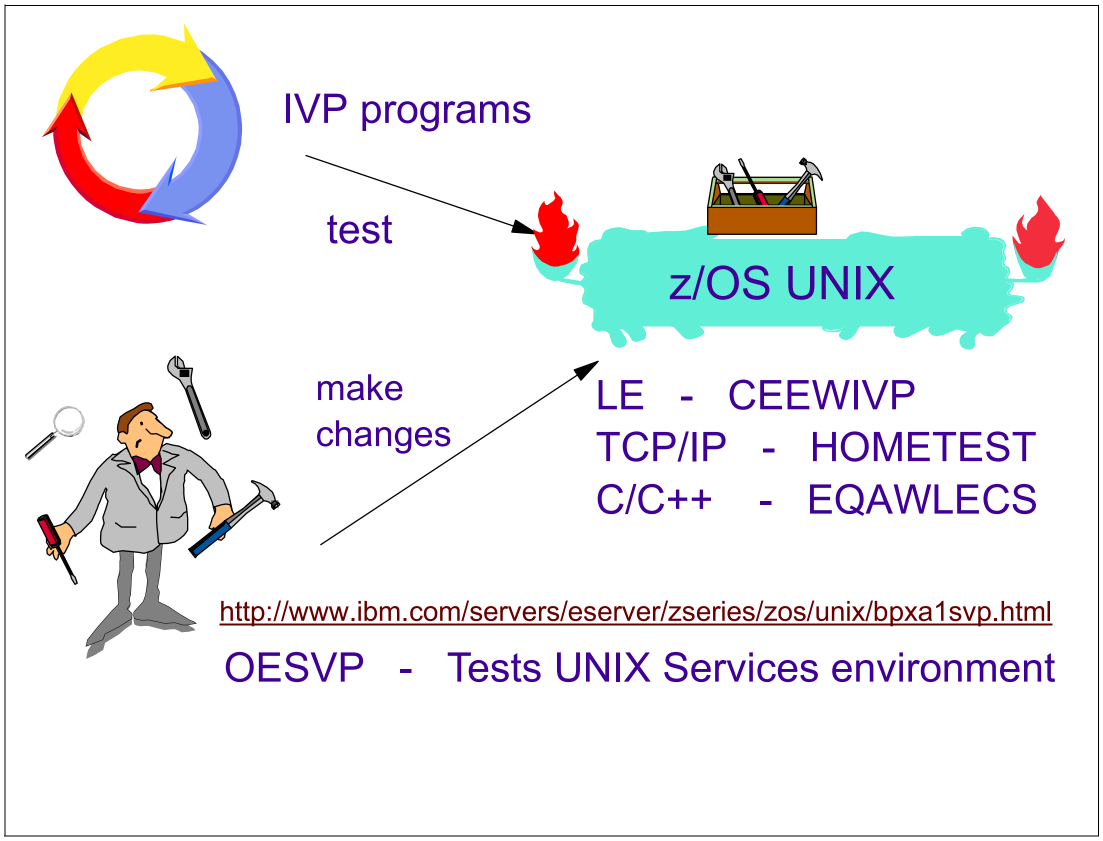

IVP programs
test
z/OS UNIX
make
changes
LE - CEEWIVP
TCP/IP - HOMETEST
C/C++ - EQAWLECS
http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1svp.html
OESVP - Tests UNIX Services environment

Figure 5-27 Customizing the installation verification program

After customizing the different tools and changing the environment, you have to test the new settings with the related Installation Verification Programs (IVP).

- LE environment

  To verify that Language Environment was installed properly, run CEEWIVP, which is found in your SCEESAMP library. Refer to the comments in the job for instructions, expected condition codes, and expected output.

- IBM Communications Server IP

  Verify your host name and address configuration with HOMETEST. This program will test the system configuration defined by the HOSTNAME, DOMAINORIGIN, and NSINTERADDR statements in the TCPIP.DATA data set.

- C/C++ with Debug Tool

  If you are using the C/C++ debug tools, you have to submit the following jobs in this order:

```
hlq.SCBCJCL(EQAWLECS)
hlq.SCBCJCL(EQAWLU62)
hlq.SCBCJCL(EQAWIVP2)
```

- C/C++ Compiler

  If C/C++ has been enabled, verify that the following C/C++ components are installed properly:

  – C/C++ Compilers

    Run the jobs HLB4801F and HLB4801G from the ServerPac Installation Dialog.

  – C/C++ Host Performance Analyzer

    In order to test the C/C++ Host Performance Analyzer you can run the two jobs HLB4801M and HLB4801N from the ServerPac Installation Dialog, or find the same JCL in hlq.SCTVJCL with the names CTVFINT and CTVFUNK.

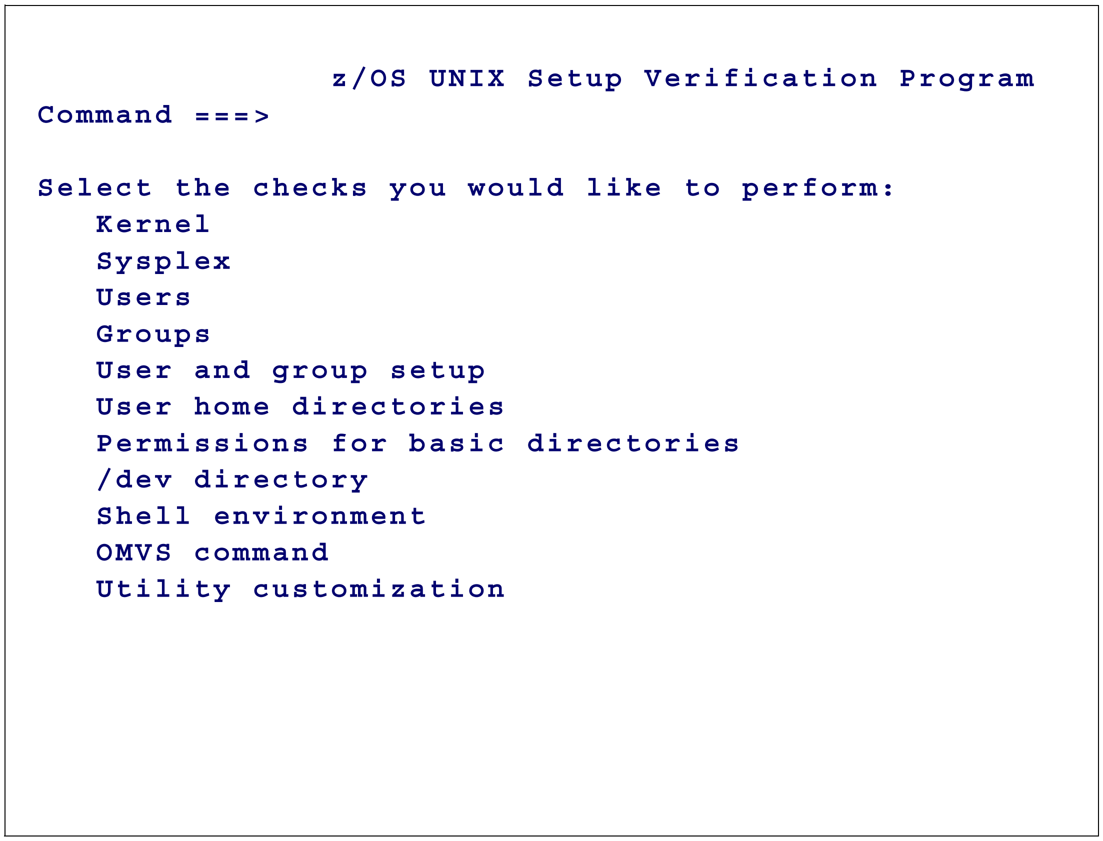
  – C/C++ IBM Open Class® Library

    To test the Open Class Library, use the jobs HTV4821R, HTV4821S and HTV4821T, which can be run from the ServerPac Installation dialog or from the hlq.SCLBJCL library with following names: CLB3JIV1, CLB3JIV2, and CLB3JIV3.

- z/OS UNIX environment and tools

  To test the z/OS UNIX environment and tools use the program **oesvp**, which can be downloaded from http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1svp.html with the name oesvp.exec.bin.

5.28 Installation Verification Program (IVP)



```
               z/OS UNIX Setup Verification Program
Command ===>

Select the checks you would like to perform:
   Kernel
   Sysplex
   Users
   Groups
   User and group setup
   User home directories
   Permissions for basic directories
   /dev directory
   Shell environment
   OMVS command
   Utility customization
```

Figure 5-28 Using the installation verification program

The IVP for z/OS UNIX ISPF panels is shown Figure 5-28. During the z/OS Shell test, do not interrupt the process when the indicator RUNNING/INPUT changes; instead, press PF10.

This utility requires ISPF version 4.1 or higher, as well as MVS 5.2.2 or higher.

For more information, see *ServerPac: Installing Your Order* or the ServerPac Program Directory.

The Setup Verification Program (SVP) lets you check for troublesome setup errors before they trip you up. After you have followed the instructions in *z/OS UNIX Planning*, GA22-7800, and completed your setup and customization (including the z/OS shell and utilities), you can run the SVP.

Using the SVP, you can:

- Verify that each user has a UID and OMVS segment defined, and each group has a GID.
- Check for duplicate assignment of UIDs and GIDs.
- Verify that each user has access to and owns a home directory and has read, write, and search access to it.
- Check the permissions for several directories usually set up at installation.
- Check that files in the /dev directory are defined correctly. Reconcile the number of pseudo-ttys and file descriptor files with the BPXPRMxx definitions. On Release 7 and

above, it does minimal checking in the /dev directory because the files are created by the system as needed.

- Verify that the z/OS shell will run.
- Verify that the `OMVS` command will run.
- Check customization for utilities. The program checks:
  - Files that have been copied from /samples to /etc
  - Terminfo files
  - Settings for some environment variables
  - Ability to compile and run a program
  - Various other utility customizations
- Verify, if SYSPLEX(YES) is specified, that the root directory has the correct symlinks defined. Sysplex support for the shared HFS is available with Release 9.

If it detects a problem, the SVP warns you about it and, if you request, corrects the problem. We estimate the SVP can take up to one-half hour to complete; but the exact amount of time depends on your system.

To use the SVP:

- You must be a superuser (UID=0) with RACF SPECIAL authority, or the equivalent.
- Your system must be at MVS release 5.2.2 or higher, or any release of z/OS.
- Your system must be at ISPF version 4.1 or higher.
- You can use any security product; RACF is not required.

6

Security customization

This chapter provides an overview of UNIX System Services security. It provides information on how to customize the security definitions for UNIX System Services. In addition to introducing the basic concepts of UNIX security, this chapter provides the details on how to:

- Define new UNIX System Services users and groups
- Change existing users and groups for UNIX System Services
- Set up security for the UNIX System Services kernel
- Set up security for UNIX System Services daemons
- Set up security for UNIX System Services products
- Define users and groups to RACF
- Manage superusers
- Define permissions bits and how they are used
- Define RACF profiles for OpenEdition
- Protect the daemons programs
- Understand server security
- Utilize various auditing capabilities

The security configuration is an important prerequisite to enabling z/OS UNIX System Services. At least minimal security work is required just to make z/OS UNIX start. After that, the security configuration becomes increasingly more complex as users are defined and workload is brought onto the system. This chapter gives you a comprehensive view of the work required to set up z/OS UNIX in a way that is secure and in line with the needs of typical business organizations.

6.1 RACF OMVS segments

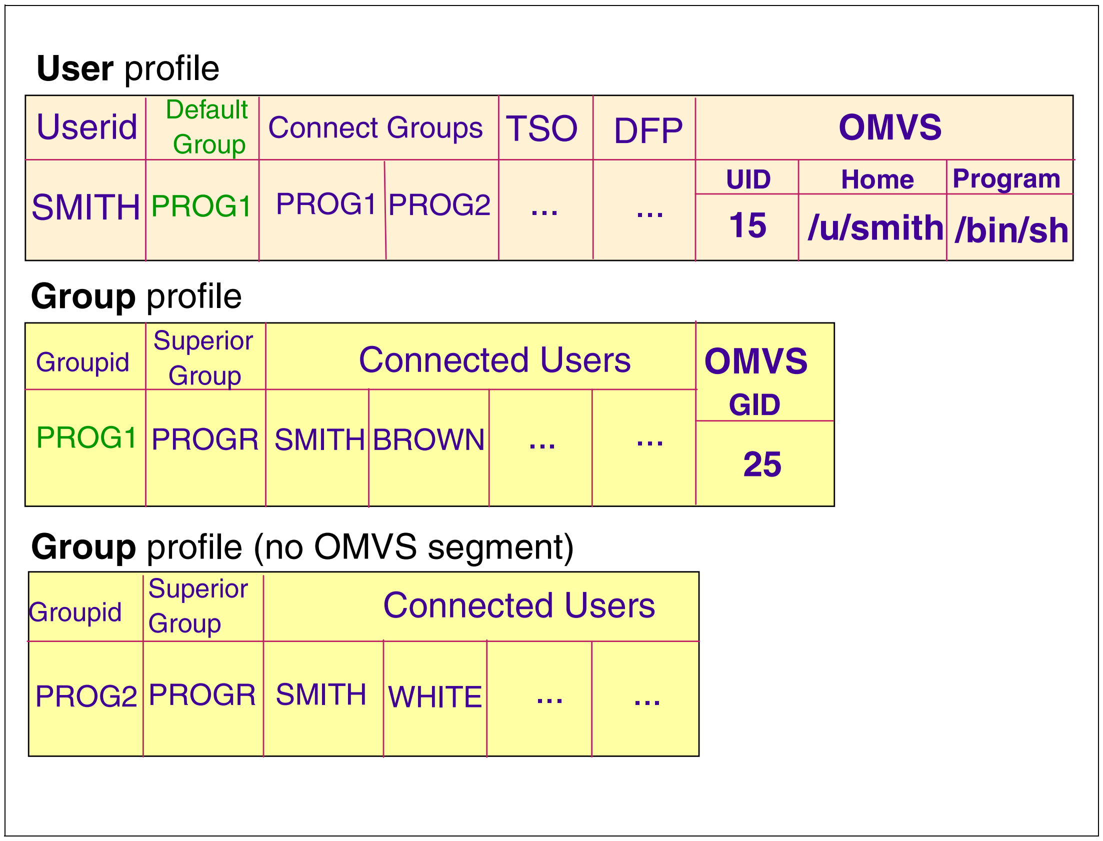


Figure 6-1 Defining RACF OMVS segments

The RACF user profile definition was expanded with a segment called OMVS for z/OS UNIX support. All users and programs that need access to z/OS UNIX must have a RACF user profile defined with an OMVS segment which has, as a minimum, a UID specified. A user without a UID cannot access z/OS UNIX.

This segment has three fields:

UID	A number from 0 to 2147483647 that identifies a z/OS UNIX user. A z/OS UNIX user must have a UID defined.
HOME	The name of a directory in the file system. This directory is called the home directory and becomes the current directory when the user accesses z/OS UNIX. This field is optional. The home directory is the current directory when a user invokes z/OS UNIX. During z/OS UNIX processing, this can be changed temporarily by using the `cd` (change directory) shell command. The command will not change the value in the RACF profile. The directory specified as home directory in the RACF profile must exist (be pre-allocated) before a user can invoke z/OS UNIX. If a home directory is not specified in RACF, the root (/) directory will be used as default.
PROGRAM	The name of a program. This is the program that will be started for the user when the user begins a z/OS UNIX session. Usually this is the program name for the z/OS UNIX shell. This field is optional.

The RACF group also has a segment called OMVS to define z/OS UNIX groups. It contains only one field:

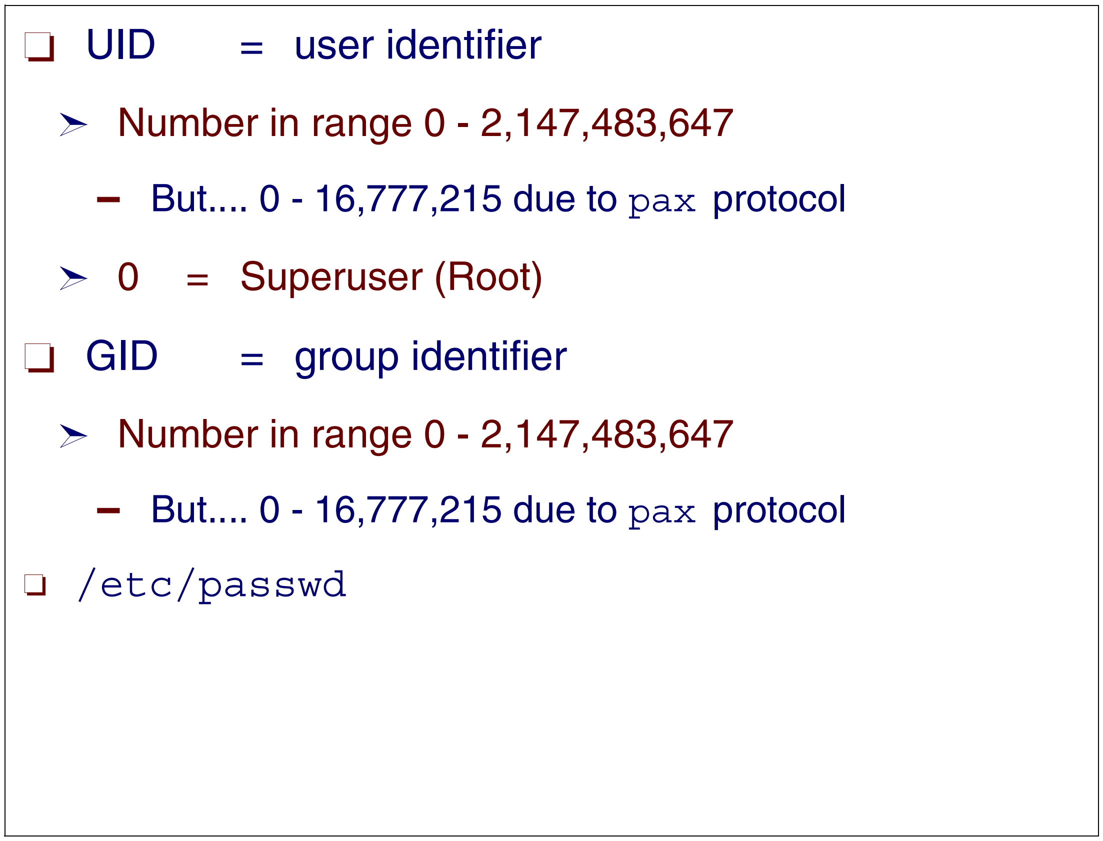
GID	A number from 0 to 2147483647 that identifies a z/OS UNIX group.

The example in Figure 6-1 shows a user profile for TSO/E user ID SMITH which is connected to two groups, PROG1 and PROG2. SMITH is defined as a z/OS UNIX user because he has a UID specified. His home directory is /u/smith and he will get into the shell when he issues the `OMVS` command because the name of the shell, /bin/sh is specified as the program name.

A program that will access z/OS UNIX and run as a started task (for example, RMFGAT) or a daemon (for example, the inetd daemon, which is used for remote login (rlogin) to the shell via TCP/IP) must also be defined to RACF with a user profile and a UID specified. This type of user does not require a home directory or a program specified in the OMVS segment. The home directory and program are important for people's user IDs.

The RACF profile for a group is also extended with an OMVS segment. A z/OS UNIX group is a RACF group with a GID specified in the OMVS segment. The figure shows that group PROG1 is also a z/OS UNIX group with a GID value of 25. The group PROG2 does not have an OMVS segment and therefore is not a z/OS UNIX group.

6.2 UNIX security



- UID = user identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol
  - 0 = Superuser (Root)
- GID = group identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol
- `/etc/passwd`

Figure 6-2 UNIX security

UNIX systems have a concept of users and groups similar to RACF. A user identifier (or UID) is a number between 0 and some large number that varies between brands of UNIX. User numbers do not have to be unique and it is possible (though not recommended) for several users to share the same UID, and even be logged on at the same time. UNIX sees these users as being the same entities and they receive the same levels of authorization. A user with UID=0 is what is called a superuser (“root” on some brands of UNIX). A superuser has unlimited authority within a UNIX environment. For obvious reasons, UID=0 needs to be strictly controlled.

Users are all related to a group. Groups allow authority to be controlled in a more economical way, in that giving access to a group is a lot easier than giving access to a few hundred users individually. A group identifier (or GID) is a number between 0 and some large number that varies between brands of UNIX. Any number of users can share the same GID.

The z/OS UNIX implementation of UNIX allows UID and GID numbers up to 2,147,483,647, but due to restrictions in UNIX design, it is recommended that the maximum number used be no more than 77,777,777.

UNIX systems typically store their user/group information in a file called /etc/passwd. This file does not exist under z/OS UNIX, because RACF (or a like security product) is used instead.

6.3 z/OS UNIX users and groups

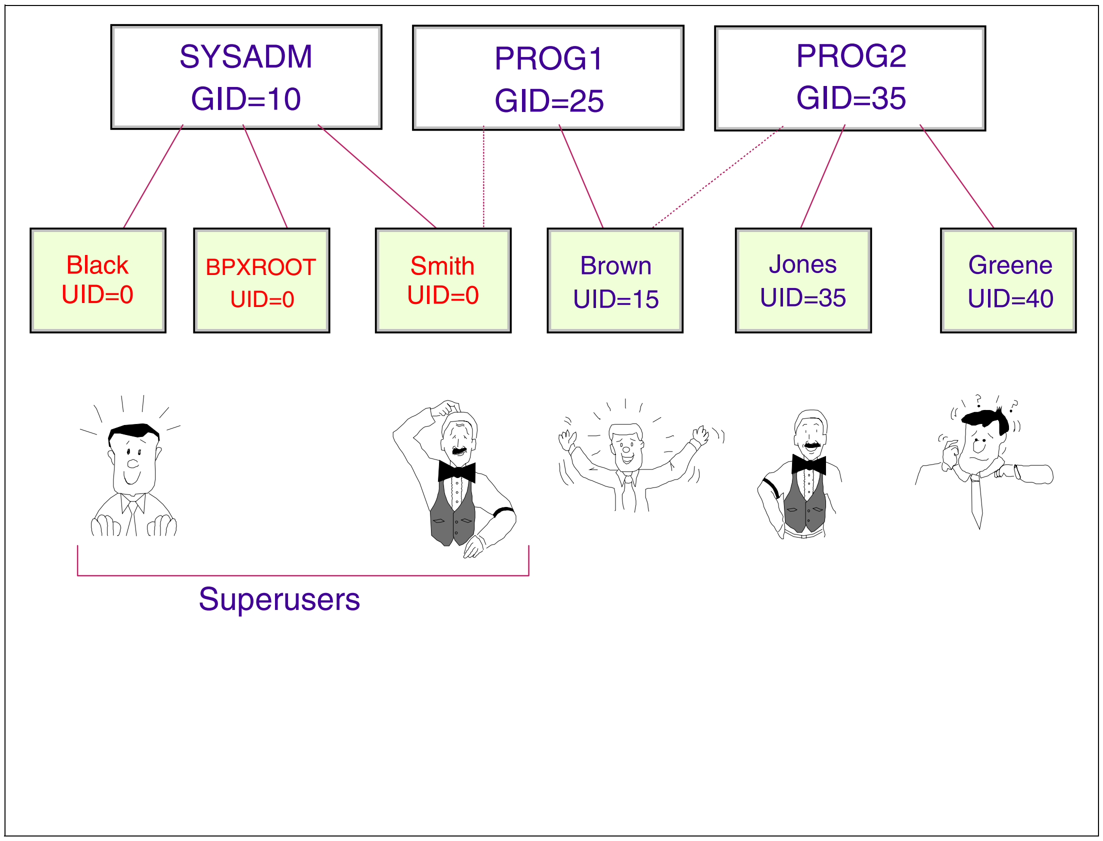


Figure 6-3 Defining z/OS UNIX users and groups

z/OS UNIX users are TSO/E user IDs with a RACF segment called OMVS defined for z/OS UNIX use. All users that want to use z/OS UNIX services must be defined as z/OS UNIX users. Similar to users in a UNIX system, z/OS UNIX users are identified by a UID (user identification). The UID has a numerical value.

There are two types of users:

- User (regular user)
  - Identified by a non-zero UID.
- Superuser (authorized/privileged user). A superuser can be any of the following:
  - A z/OS UNIX user with a UID=0.
  - A started procedure with a trusted or privileged attribute in the RACF started procedures table.

The concept of superuser comes from UNIX. Sometimes it is also referred to as *root* authority. A superuser can:

- Pass all z/OS UNIX security checks, so that the superuser can access any file in the hierarchical file system. A superuser does not get any additional authorities to access MVS/ESA resources. The authority is limited to the z/OS UNIX component.
- Manage z/OS UNIX processes and files.

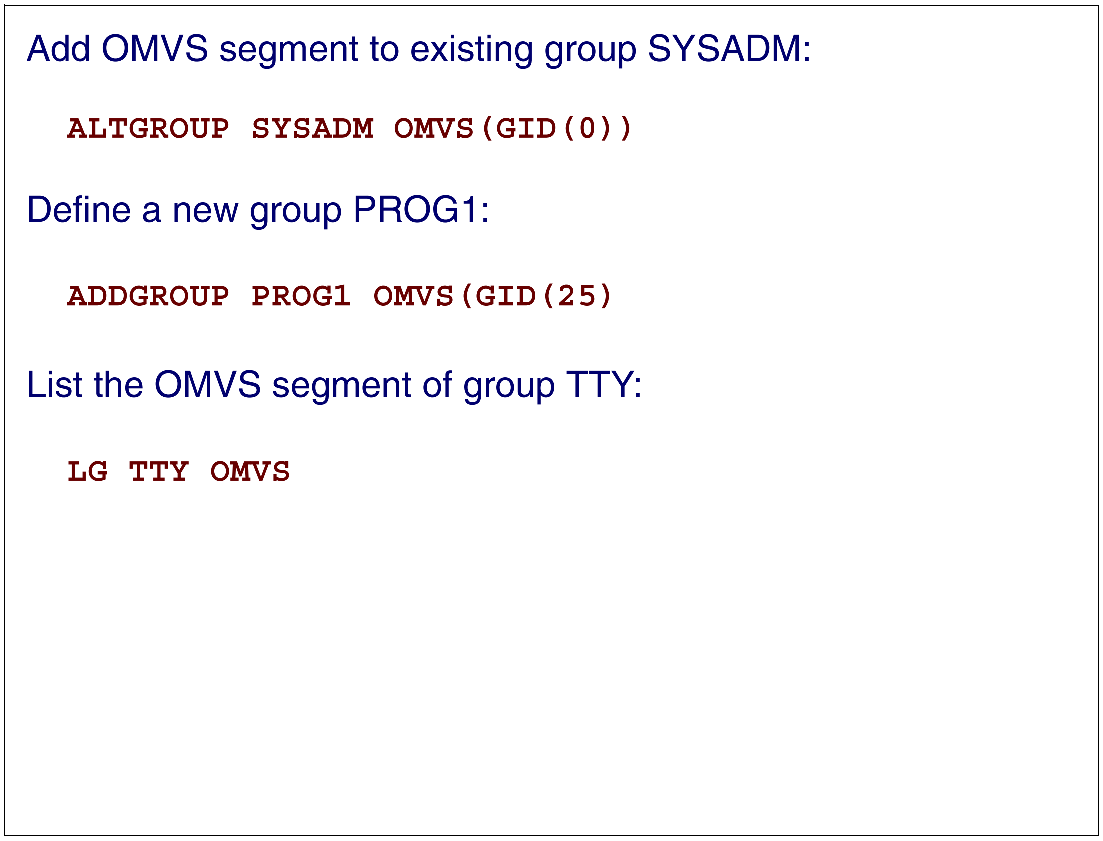
- Have an unlimited number of processes running concurrently. For a started procedure, this is true only if it has a UID of 0. It is not true for a trusted or privileged process with a different UID.
- Change identity from one UID to another.
- Use **`setrlimit`** to increase any of the system limits for a process.

A superuser is usually a system administrator, or it can be a started procedure which is authorized by the RACF started procedures table or the RACF STARTED class.

z/OS UNIX users belong to one or more groups in the same way that TSO/E users belong to groups. A z/OS UNIX group is a RACF group with a GID defined. The GID has a numerical value.

Reminder: Multiple users may have the same UID.

6.4 RACF commands to define groups



Add OMVS segment to existing group SYSADM:

```
ALTGROUP SYSADM OMVS(GID(0))
```

Define a new group PROG1:

```
ADDGROUP PROG1 OMVS(GID(25)
```

List the OMVS segment of group TTY:

```
LG TTY OMVS
```

Figure 6-4 RACF commands used to define z/OS UNIX groups

The **ALTGROUP**, **ADDGROUP**, and **LISTGRP** commands have keywords for administering the OMVS segment.

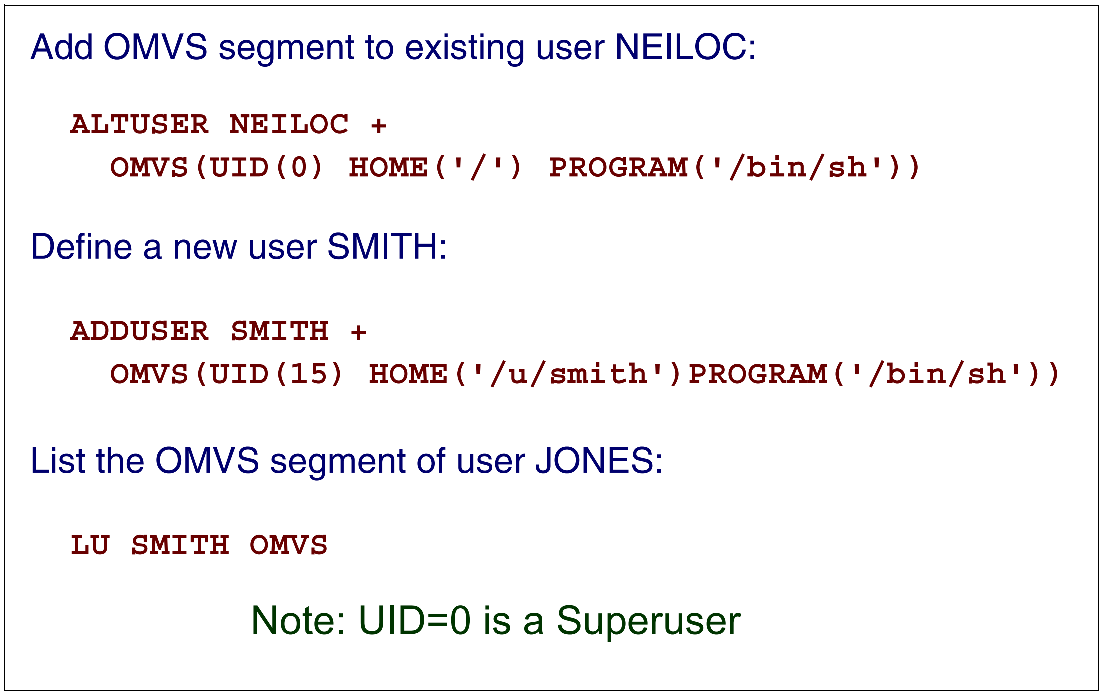
- Use **ALTGROUP** (**ALG**) to modify an existing RACF group, with or without an OMVS segment. The OMVS segment can be added, modified, or deleted.
- Use **ADDGROUP** (**AG**) to define a new RACF group, with or without an OMVS segment. The OMVS keyword can be used to define this group as a z/OS UNIX group.
- The **LISTGRP** (**LG**) command will display the OMVS segment if the OMVS keyword is specified.

A z/OS UNIX group is a RACF group with an OMVS segment and a GID defined. Figure 6-4 shows examples of how to use the RACF commands to add, change or delete the OMVS segment for a group.

Using `NOGID` will remove a GID definition, and `NOOMVS` will remove the OMVS segment.

A user can belong to (or be connected to) multiple groups. A z/OS UNIX user must belong to at least one z/OS UNIX group. It is not necessary that all the groups a z/OS UNIX user belongs to are defined as z/OS UNIX groups. Only the groups that are defined as z/OS UNIX groups will be used for authorization checking in z/OS UNIX.

6.5 RACF commands to define users



Add OMVS segment to existing user NEILOC:

```
ALTUSER NEILOC +
  OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

Define a new user SMITH:

```
ADDUSER SMITH +
  OMVS(UID(15) HOME('/u/smith')PROGRAM('/bin/sh'))
```

List the OMVS segment of user JONES:

```
LU SMITH OMVS
```

Note: UID=0 is a Superuser

Figure 6-5 RACF commands to define z/OS UNIX users

The RACF commands **ALTUSER**, **ADDUSER** and **LISTUSER** have keywords for administering the OMVS segment, as follows:

- Use the **ALTUSER** (**ALU**) command to change the definitions for an existing TSO/E user ID. An OMVS segment can be added, modified, or deleted.
- Use the **ADDUSER** (**AU**) command to define a new TSO/E user ID with or without an OMVS segment.
- The **LISTUSER** (**LU**) command is used for listing the definitions for a TSO/E user ID. When the OMVS keyword is specified, the values specified in the OMVS segment will also be listed.

Note: The values shown are case-sensitive in the definitions for HOME and PROGRAM.

ADDUSER sets up a new user, whereas **ALTUSER** modifies an existing user. To remove a specification in the OMVS segment, use the keywords `NOUID` (to remove UID), `NOHOME` (to remove home directory definition), or `NOPROGRAM` (to remove program definition). The `NOOMVS` keyword will remove all the definitions in the OMVS segment.

The OMVS keyword must be used on the **LISTUSER** command if you want the OMVS segment definitions to be displayed.

The z/OS UNIX ISPF shell (ISHELL) provides some menus for administering user IDs. However, before a user can use the ISHELL, he must have a user ID with a UID defined (and a group with a GID defined). This must be done using the RACF commands.

6.6 LU and LG command examples

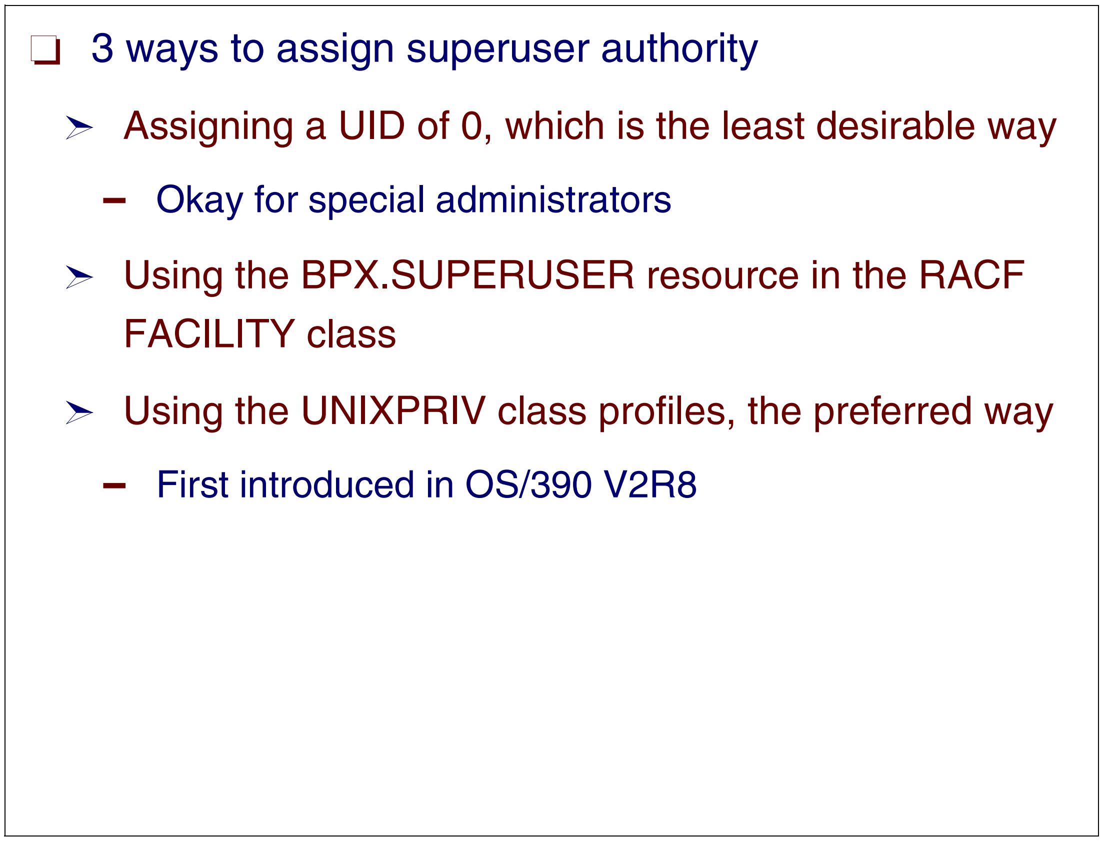
```
lu smith omvs noracf

  USER=SMITH
  OMVS INFORMATION
  ----------------
  UID= 0000000015
  HOME= /u/smith
  PROGRAM= /bin/sh

lg prog1 omvs noracf

  INFORMATION FOR GROUP PROG1
  OMVS INFORMATION
  ----------------
  GID= 0000000025
```

Figure 6-6 RACF commands to display z/OS UNIX OMVS segments

Specifying the `NORACF` option stops the `LU` and `LG` general information from being displayed.

Note that a user will only be allowed to display their OMVS segments if RACF Field-Level Access has been enabled.

6.7 Superuser with appropriate authority



- 3 ways to assign superuser authority
  - Assigning a UID of 0, which is the least desirable way
    - Okay for special administrators
  - Using the BPX.SUPERUSER resource in the RACF FACILITY class
  - Using the UNIXPRIV class profiles, the preferred way
    - First introduced in OS/390 V2R8

Figure 6-7 Defining superuser authority

When you are defining users, you might want to define some of them with appropriate superuser privileges. There are three ways of assigning superuser authority, as follows:

- Assigning a UID of 0, which is the least desirable way
- Using the BPX.SUPERUSER resource in the FACILITY class
- Using the UNIXPRIV class profiles, the preferred way

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 user IDs (as opposed to started procedures) with a UID(0) superuser authority.

To summarize the choices, UID(0) gives you access to all UNIX functions and resources, as is true for all UNIX systems.

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.

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.

6.8 z/OS UNIX-level security for superusers

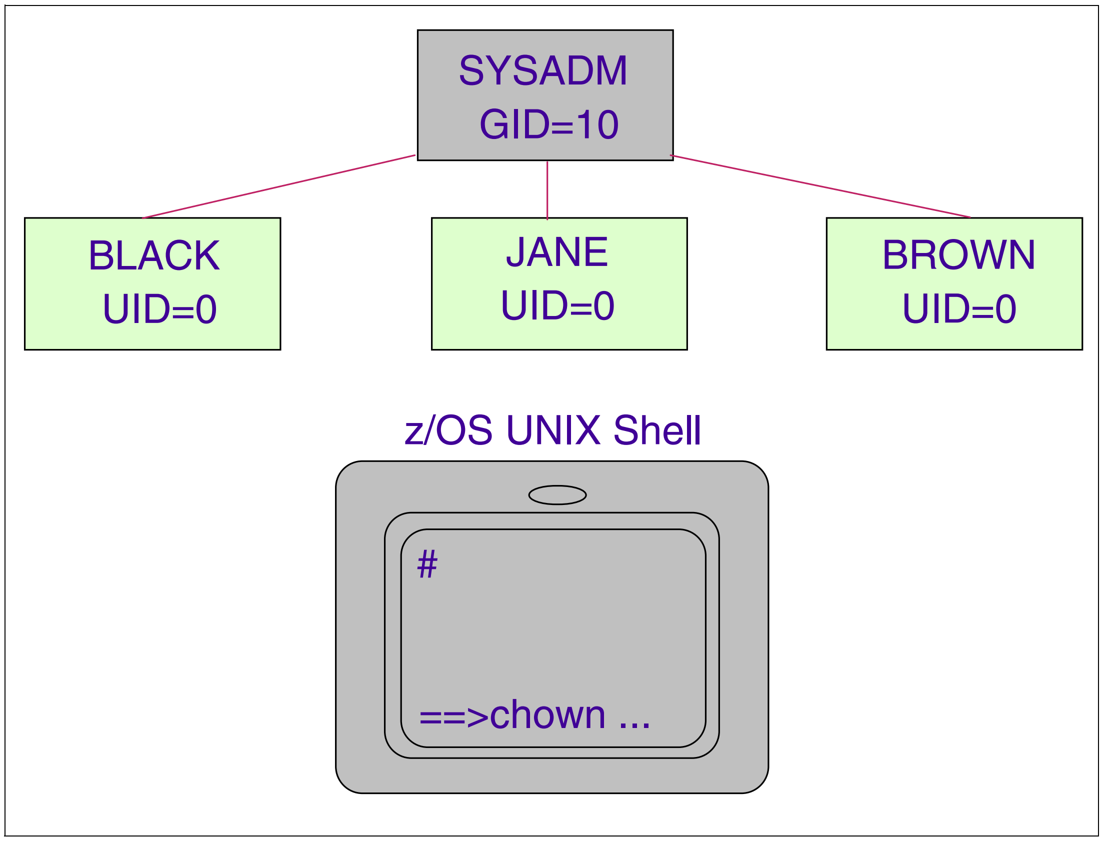


Figure 6-8 Defining superusers

Superusers are special users in a z/OS UNIX environment and they are identified by a UID value of 0. One way of defining superusers is to set the UID to 0 in the user's OMVS segment. Using this method, the user always runs as a superuser. Multiple users can be defined with a UID of 0.

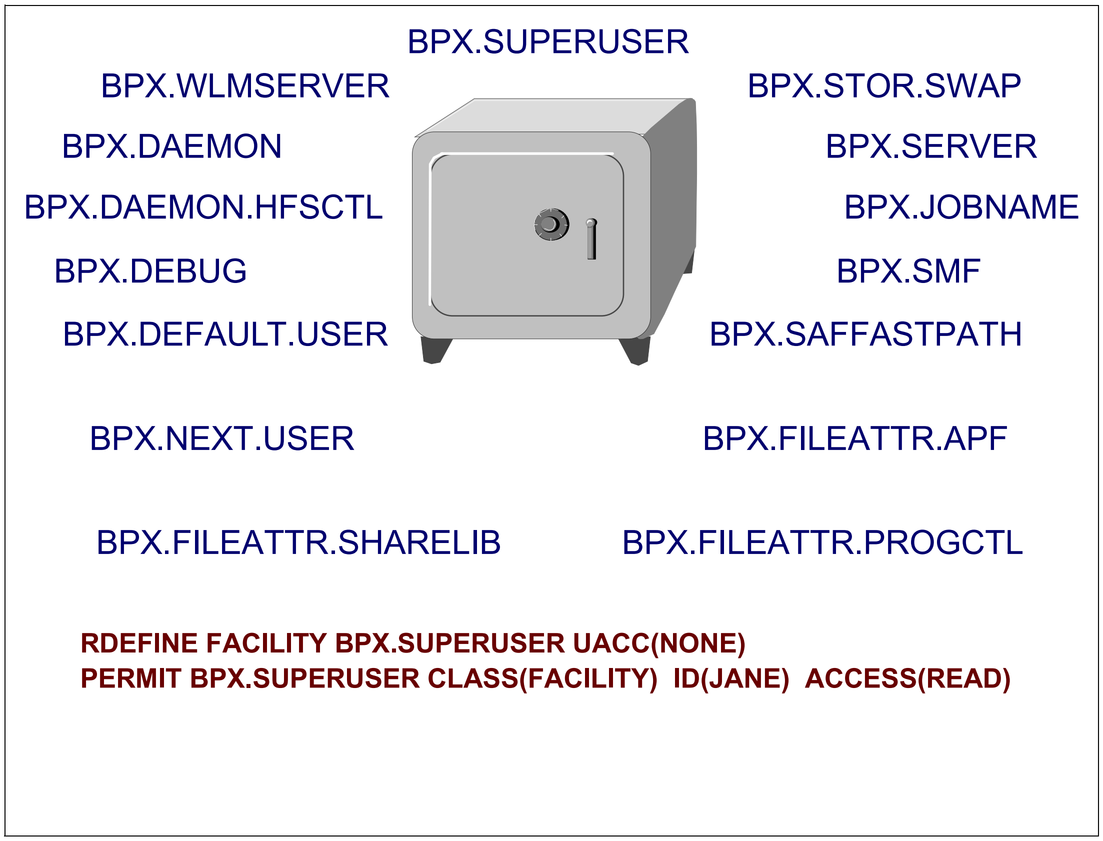
Define some system administrators as superusers by adding an OMVS segment to their user profile. Define the home directory as root (/). Add an OMVS segment to the group(s) which these users are connected to.

Superuser status is not related to being in supervisor state, PSW key 0, or using APF-authorized instructions, macros, or callable services.

`chown` Command

This command is used to change the owner or group of a file or directory. **`chown`** sets the user ID to *owner* for the files and directories named by pathname arguments. Owner can be a user name from the user database, or it can be a numeric user ID. If a numeric owner exists as a user name in the user database, the user ID number associated with that user name is used.

6.9 z/OS UNIX security



```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
PERMIT BPX.SUPERUSER CLASS(FACILITY)  ID(JANE)  ACCESS(READ)
```

Figure 6-9 z/OS UNIX RACF profiles for security

RACF Facility class profile example:

```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(JANE) ACCESS(READ)
```

Following are a number of FACILITY class profiles that you may want to define for z/OS UNIX:

- BPX.DAEMON

  BPX.DAEMON serves two functions in the z/OS UNIX environment:

  – Any superuser 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. This identity change can only occur if the target user ID has an OMVS segment defined.

    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 loaded into an address space that requires daemon level authority must be defined to program control. If the BPX.DAEMON 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
  - _password()

- BPX.DAEMON.HFSCTL

  Controls which users with daemon authority are allowed to load uncontrolled programs from MVS libraries into their address space.

- BPX.DEBUG

  Users with read access to the BPX.DEBUG FACILITY class profile can use **ptrace** (via dbx) to debug programs that run with APF authority or with BPX.SERVER authority.

- BPX.DEFAULT.USER

  Not all users and groups need to have discrete OMVS segments defined for them. For example:

  – Users who need to use sockets and do not need any other UNIX services. In the past, users could open sockets without any other special permissions.

  – Users who want to run multithreading PL/1 programs. PL1 uses some kernel services, so the OMVS segments are currently required to get dubbed.

  – Users who just want to experiment with the shell and do not have an OMVS segment defined.

  The default OMVS segments will reside in a USER profile and a GROUP profile. The names of these profiles are selected by the installation, and stored in the application data field of BPX.DEFAULT.USER.

- BPX.FILEATTR.APF

  Controls which users are allowed to set the APF-authorized attribute in an HFS 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-controlled attribute in an HFS file. Programs marked with this attribute can execute in server address spaces that run with a high level of authority.

- BPX.FILEATTR.SHARELIB

  Indicates that extra privilege is required when setting the shared library extended attribute via the chattr() callable service. This prevents the shared library region from being misused.

- 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.NEXT.USER

  Enables automatic assignment of UIDs and GIDs. The APPLDATA of this profile specifies a starting value, or range of values, from which RACF will derive unused UID and GID values.

- BPX.SAFFASTPATH

  Enables faster security checks for file system and IPC constructs.

- 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
- _password()

- BPX.SMF

  Checks if the caller attempting to cut an SMF record is allowed to write an SMF record or test if an SMF type or subtype is being recorded.

- BPX.SRV.userid

  Allows users to change their UID if they have access to BPX.SRV.userid, where uuuuuuuu is the MVS user ID associated with the target UID. BPX.SRV.userid is a RACF SURROGAT FACILITY class profile.

- BPX.STOR.SWAP

  Controls which users can make address spaces nonswappable. Users permitted with at least READ access to BPX.STOR.SWAP can invoke the **`_mlockall()`** function to make their address space either nonswappable or swappable.

  When an application makes an address space nonswappable, it may 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 using this service should warn the customer about this side effect in their installation documentation.

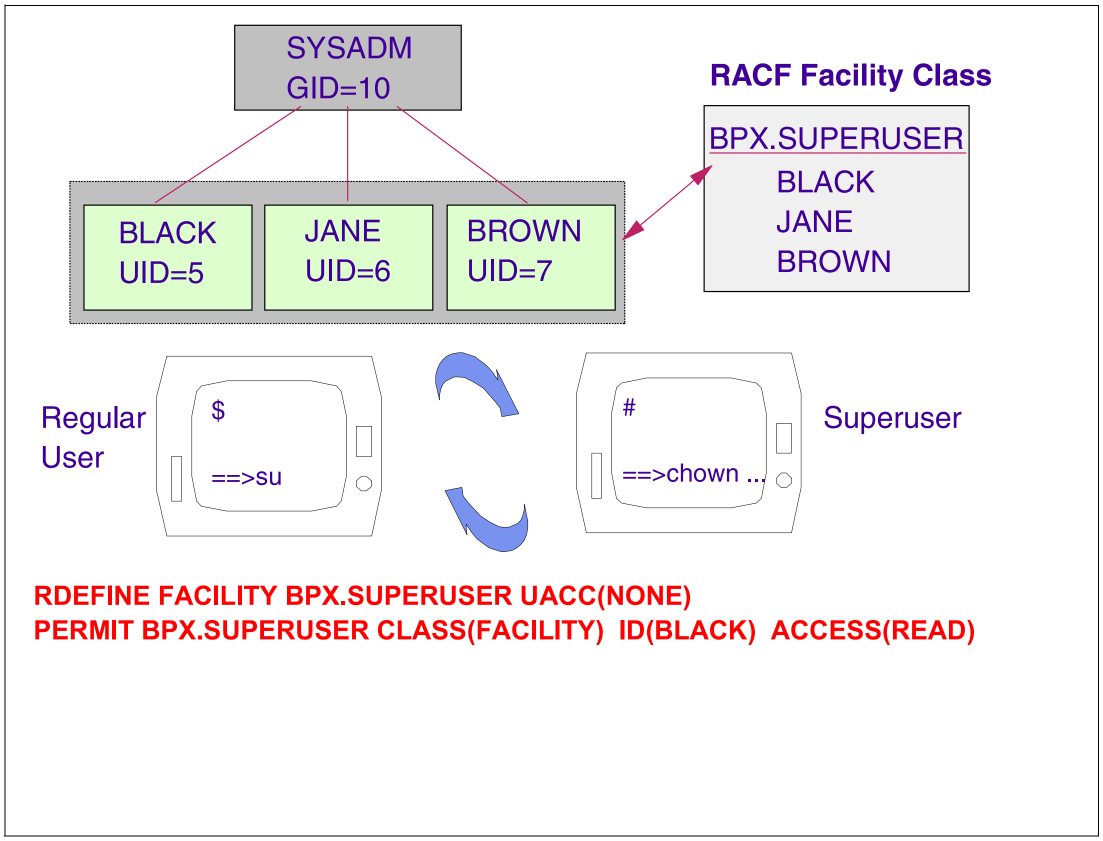
- BPX.SUPERUSER

  Users with access to the BPX.SUPERUSER FACILITY class profile can switch to superuser authority (effective UID of 0).

- BPX.WLMSERVER

  Controls access to the WLM server functions **`_server_init()`** and **`_server_pwu()`**. A server application with read permission to this FACILITY class profile can use the server functions, as well as the WLM C language functions, to create and manage work requests.

6.10 z/OS UNIX security: BPX.SUPERUSER



```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
PERMIT BPX.SUPERUSER CLASS(FACILITY)  ID(BLACK)  ACCESS(READ)
```

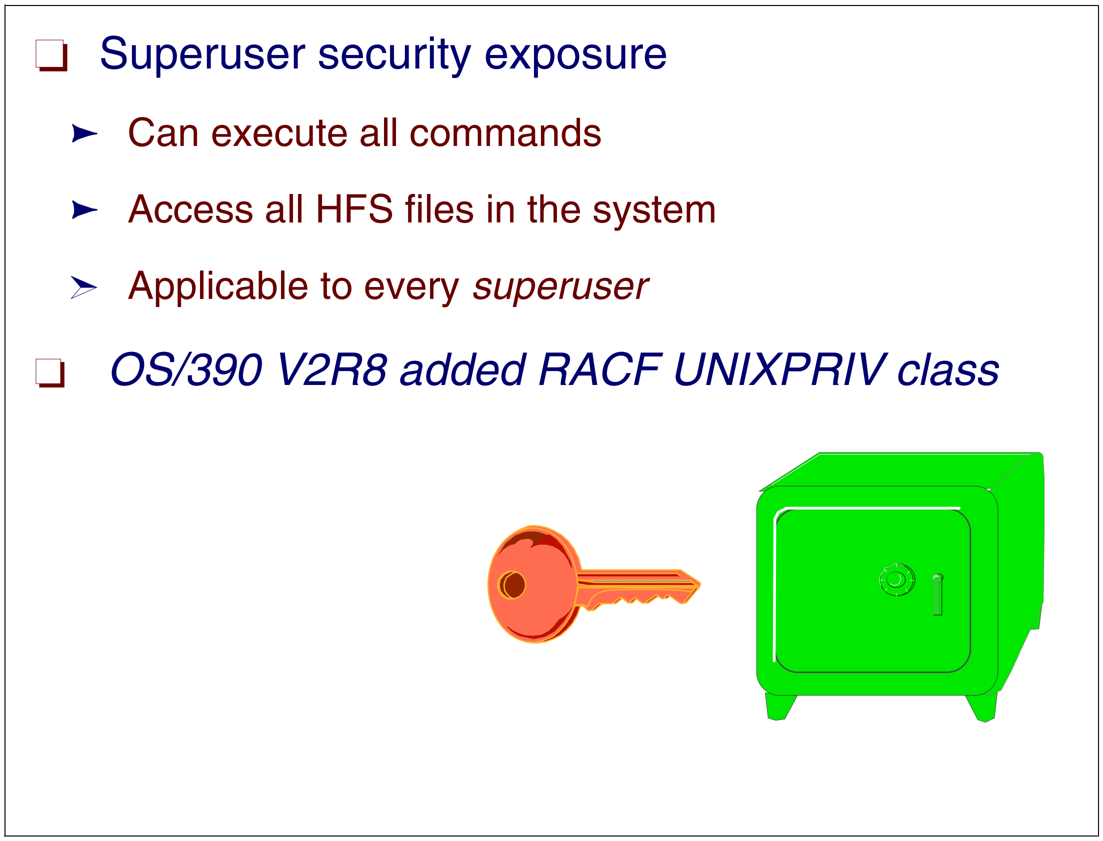
Figure 6-10 Using BPX.SUPERUSER to define superuser authority

There is an alternative way of defining superusers. Define system administrators in RACF with non-zero UIDs, and give them READ access to a RACF FACILITY class called **BPX.SUPERUSER**. Users with this authority will be able to temporarily switch to become superuser when this authority is required for administrative tasks. These users can use any of the following methods to switch to superuser:

- In the z/OS UNIX shell, use the command `su` (switch user). This command will create a subshell where the user will have superuser authority and authorized commands can be executed. When the subshell session is ended, the user will return to the first shell session as a regular user.
- Use the **`ISHELL`** command to enter the z/OS UNIX ISPF Shell. Select the option to switch to superuser state. The user will have superuser authority until the user exits the ISHELL.
- After gaining superuser authority in the ISHELL, the user can do a split screen in ISPF and enter the **`OMVS`** command. The z/OS UNIX shell that is started will inherit the superuser authority set up in the ISHELL.

Define a user ID called BPXROOT with an OMVS segment. Specify UID=0, a home directory of / (root), and the program /bin/sh. BPXROOT should not have any special permission to MVS resources. This user ID will be used in rare cases where a daemon process tries to change the identity of a process to superuser but does not know the MVS identity of the process. BPXROOT is the default name. A different name can be used, but then the installation has to change the specification in the BPXPRMxx parmlib member: SUPERUSER(userid).

6.11 z/OS UNIX superuser granularity



- Superuser security exposure
  - Can execute all commands
  - Access all HFS files in the system
  - Applicable to every *superuser*
- *OS/390 V2R8 added RACF UNIXPRIV class*

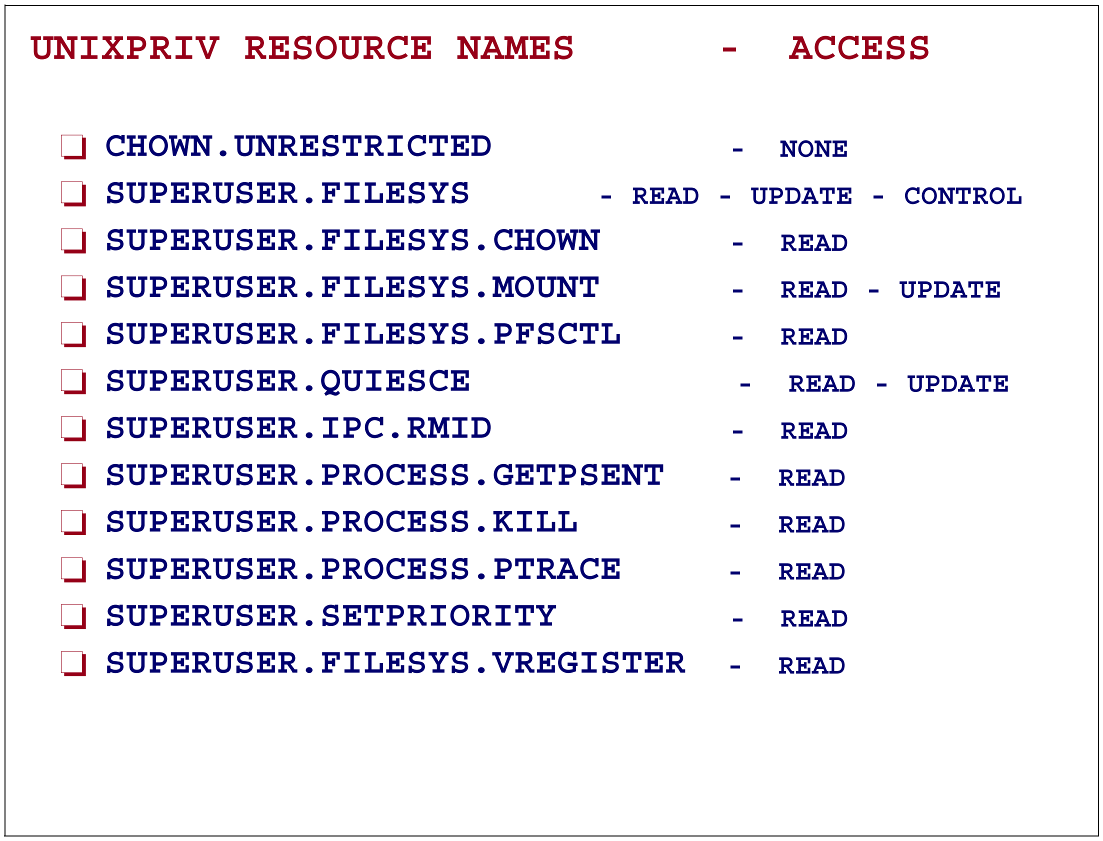
Figure 6-11 Adding superuser granularity for access

A superuser can:

- Pass all security checks, so that the superuser can access any file in the file system.
- Manage processes.
- Have an unlimited number of processes 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.
- Change identity from one UID to another.
- Use setrlimit to increase any of the system limits for a process.

You may choose to assign the UID of 0 to multiple RACF user IDs. However, you should seek to minimize the assignment of superuser authority in your installation. You can accomplish this by setting z/OS UNIX user limits and by managing superuser privileges through UNIXPRIV profiles.

6.12 Resource names: UNIXPRIV (V2R8)



UNIXPRIV RESOURCE NAMES - ACCESS

❑ CHOWN.UNRESTRICTED	- NONE
❑ SUPERUSER.FILESYS	- READ - UPDATE - CONTROL
❑ SUPERUSER.FILESYS.CHOWN	- READ
❑ SUPERUSER.FILESYS.MOUNT	- READ - UPDATE
❑ SUPERUSER.FILESYS.PFSCTL	- READ
❑ SUPERUSER.QUIESCE	- READ - UPDATE
❑ SUPERUSER.IPC.RMID	- READ
❑ SUPERUSER.PROCESS.GETPSENT	- READ
❑ SUPERUSER.PROCESS.KILL	- READ
❑ SUPERUSER.PROCESS.PTRACE	- READ
❑ SUPERUSER.SETPRIORITY	- READ
❑ SUPERUSER.FILESYS.VREGISTER	- READ

Figure 6-12 Profiles in the RACF UNIXPRIV class

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. These privileges are automatically granted to all users with z/OS UNIX superuser authority. 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 allows you to 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. The following profiles with SUPERUSER.xxxxxxxx are:

- CHOWN.UNRESTRICTED

  ACCESS required (NONE). Allows all users to use the **`chown`** command to transfer ownership of their own files

- FILESYS

  ACCESS required (READ). Allows user to read any HFS file, and to read or search any HFS directory.

  ACCESS required (UPDATE). Allows user to write to any HFS file, and includes privileges of READ access.

ACCESS required (CONTROL or higher). Allows user to write to any HFS directory, and includes privileges of UPDATE access.

- FILESYS.CHOWN

  ACCESS required (READ). Allows user to use the **chown** command to change ownership of any file.

- FILESYS.MOUNT

  ACCESS required (READ). Allows user to issue the TSO/E **MOUNT** command or the **mount** shell command with the nosetuid option. Also allows users to unmount a file system with the TSO/E **UNMOUNT** command or the **unmount** shell command mounted with the **nosetuid** command. Users permitted to this profile can use the **chmount** shell command to change the mount attributes of a specified file system.

  ACCESS required (UPDATE). 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 with the setuid option. Users permitted to this profile can issue the **chmount** shell command on a file system that is mounted with the setuid option.

- FILESYS.QUIESCE

  ACCESS required (READ). Allows user to issue **quiesce** and **unquiesce** commands for a file system mounted with the nosetuid option.

  ACCESS required (UPDATE). Allows user to issue **quiesce** and **unquiesce** commands for a file system mounted with the setuid option.

- FILESYS.PFSCTL

  ACCESS required (READ). Allows user to use the **pfsctl()** callable service.

- FILESYS.VREGISTER

  ACCESS required (READ). Allows a server to use the **vreg()** callable service to register as a VFS file server.

- IPC.RMID

  ACCESS required (READ). Allows user to issue the **ipcrm** command to release IPC resources.

- PROCESS.GETPSENT

  ACCESS required (READ). Allows user to use the **w_getpsent** callable service to receive data for any process.

- PROCESS.KILL

  ACCESS required (READ). Allows user to use the **kill()** callable service to send signals to any process.

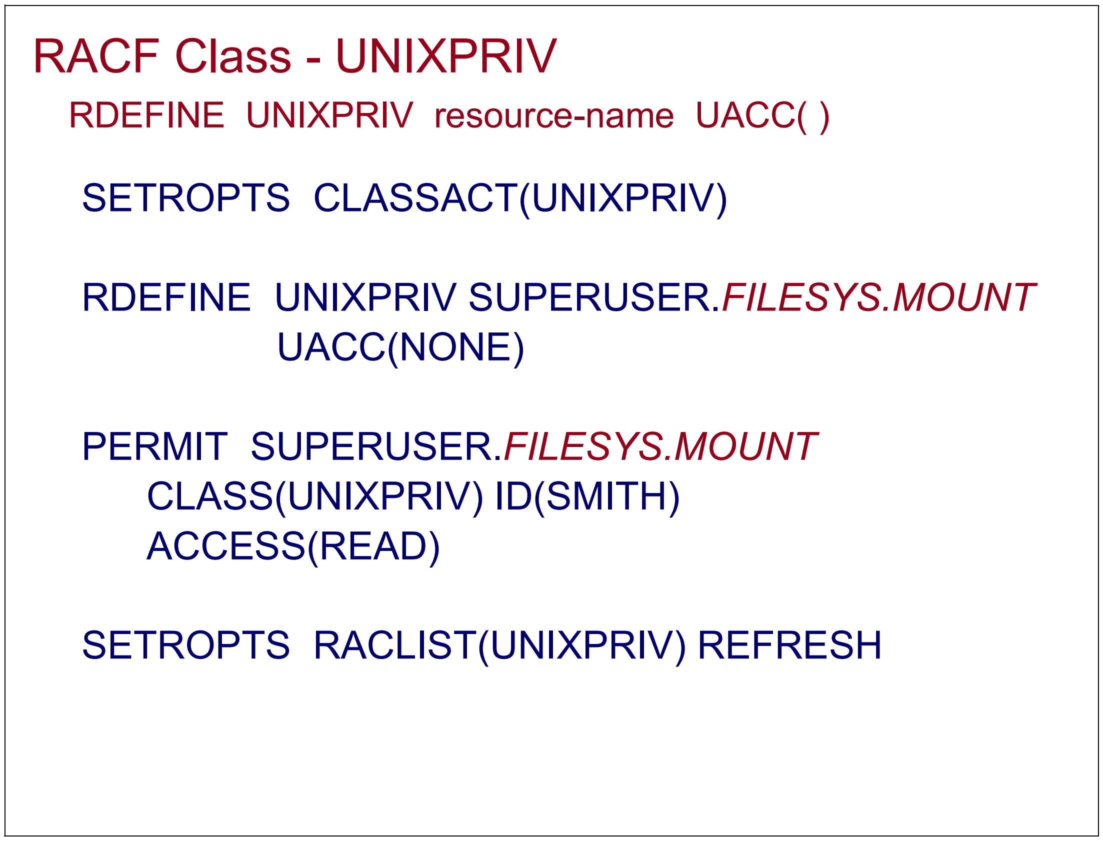
- PROCESS.PTRACE

  ACCESS required (READ). Allows user to use the **ptrace()** function through the dbx debugger to trace any process. Allows users of the **ps** command to output information on all processes. This is the default behavior of ps on most UNIX platforms.

- SETPRIORITY

  ACCESS required (READ). Allows user to increase their own priority.

6.13 z/OS UNIX superuser granularity


RACF Class - UNIXPRIV

```
RDEFINE UNIXPRIV resource-name UACC( )

SETROPTS CLASSACT(UNIXPRIV)

RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT
         UACC(NONE)

PERMIT SUPERUSER.FILESYS.MOUNT
   CLASS(UNIXPRIV) ID(SMITH)
   ACCESS(READ)

SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Figure 6-13 Defining superuser authority with the UNIXPRIV class

You can reduce the number of people who have superuser authority at you installation by defining profiles in the UNIXPRIV class that grant RACF authorization for certain z/OS UNIX privileges.

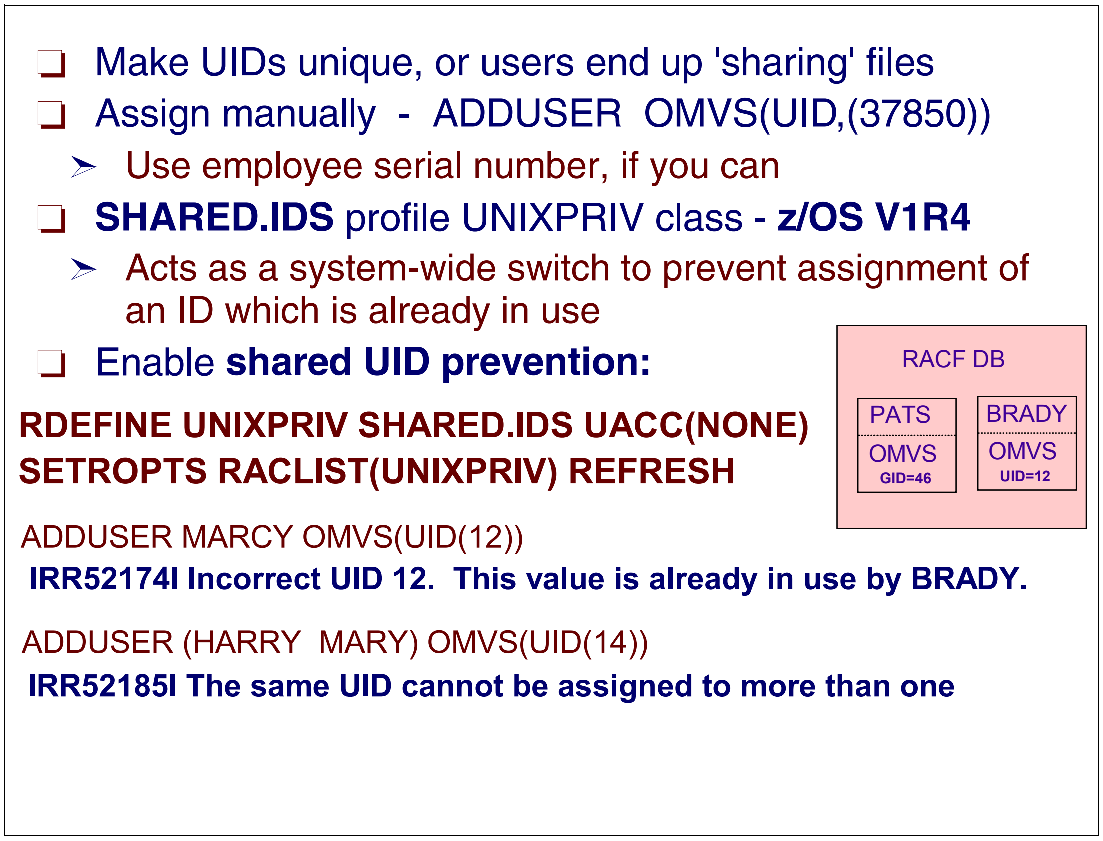
Normally, these privileges are automatically defined for all users who are defined with z/OS UNIX superuser authority. But you can use UNIXPRIV to grant certain superuser privileges, with a high degree of granularity, to users who do not have superuser authority.

For example, if users have READ access to SUPERUSER.FILESYS.MOUNT, they can issue a **mount** and **unmount** command without being a defined superuser with all superuser capabilities, as follows:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT  SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(JANE) ACCESS(READ)
SETROPTS  RACLIST(UNIXPRIV) REFRESH
```

Now user JANE (UID=35) can issue mounts, which is a superuser function. This is the only superuser function JANE can do.

6.14 Assigning UIDs



- Make UIDs unique, or users end up 'sharing' files
- Assign manually - ADDUSER OMVS(UID,(37850))
  - Use employee serial number, if you can
- **SHARED.IDS** profile UNIXPRIV class **- z/OS V1R4**
  - Acts as a system-wide switch to prevent assignment of an ID which is already in use
- Enable **shared UID prevention:**

```
RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH

ADDUSER MARCY OMVS(UID(12))
 IRR52174I Incorrect UID 12.  This value is already in use by BRADY.

ADDUSER (HARRY  MARY) OMVS(UID(14))
 IRR52185I The same UID cannot be assigned to more than one
```

Figure 6-14 Assigning UIDs with shared UID prevention

z/OS UNIX allows multiple users to have the same UID. Assigning the same UID to multiple user IDs 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.

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. This can be an employee serial number or some other unique number for each user.

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 should 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.

Shared UID prevention

In order to prevent several users from having the same UID number, a new RACF SHARED.IDS profile has been introduced in the UNIXPRIV class. This new profile acts as a system-wide switch to prevent assignment of an UID which is already in use. The use of the

SHARED.IDS profile requires AIM stage 2 or 3. To enable shared UID prevention, it is necessary to define a new SHARED.IDS profile in the UNIXPRIV class, as follows:

```
RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Once the SHARED.IDS profile has been defined and the UNIXPRIV class refreshed, it will not allow a UID to be assigned if the UID is already in use. The same is true for GIDs; it will not allow a GID to be shared between different groups.

The use of this new functionality does not affect pre-existing shared UIDs. They remain as shared once you install the new support. If you want to eliminate sharing of the same UID, you must clean them up separately. The release provides a new IRRICE report to find the shared UIDs.

6.15 Shared UID prevention

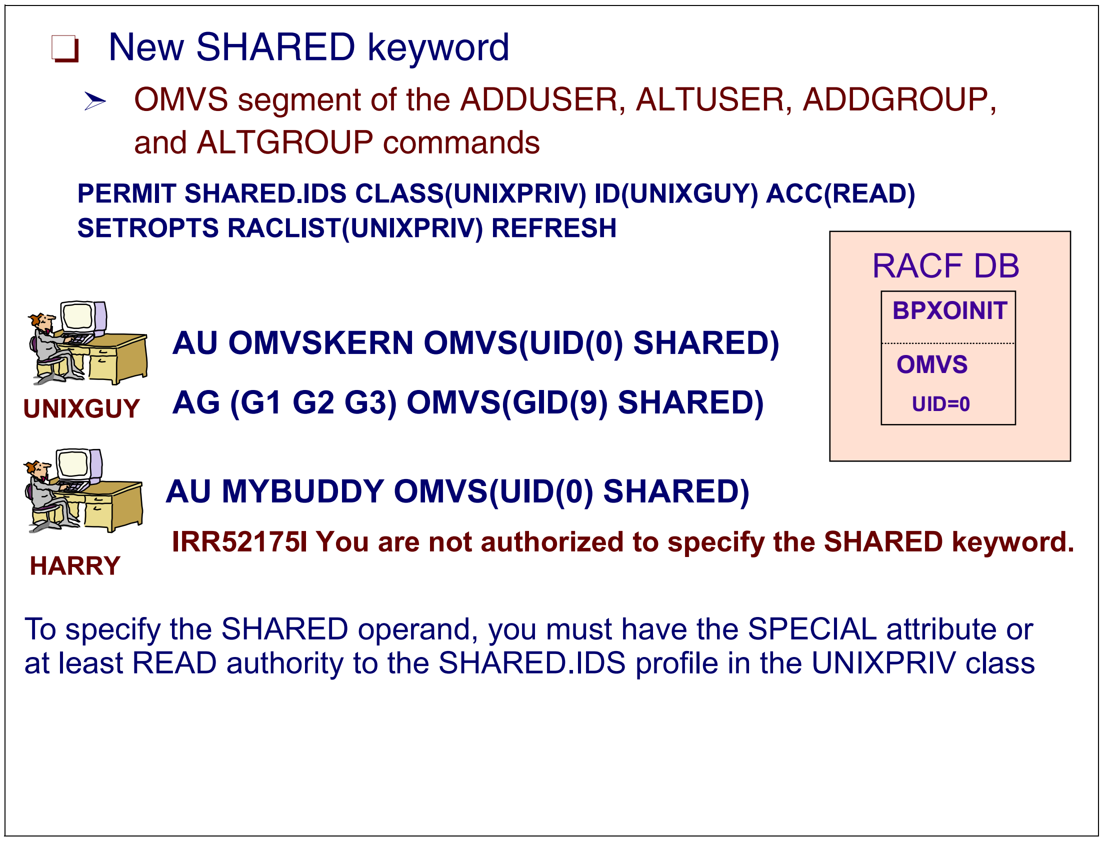

New SHARED keyword
OMVS segment of the ADDUSER, ALTUSER, ADDGROUP, and ALTGROUP commands
PERMIT SHARED.IDS CLASS(UNIXPRIV) ID(UNIXGUY) ACC(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
RACF DB
BPXOINIT
OMVS
UID=0
UNIXGUY
AU OMVSKERN OMVS(UID(0) SHARED)
AG (G1 G2 G3) OMVS(GID(9) SHARED)
HARRY
AU MYBUDDY OMVS(UID(0) SHARED)
IRR52175I You are not authorized to specify the SHARED keyword.
To specify the SHARED operand, you must have the SPECIAL attribute or at least READ authority to the SHARED.IDS profile in the UNIXPRIV class

Figure 6-15 Using the SHARED keyword to allow duplicate UID assignment

You may want to assign the same UID to multiple user IDs if these user IDs are used by the same person or persons. It may 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.

Even if the SHARED.IDS profile is defined, you may still require some UIDs to be shared and others not to be shared. For example, you may require multiple superusers with a UID(0). It is possible to do this using the new SHARED keyword in the OMVS segment of the **ADDUSER**, **ALTUSER**, **ADDGROUP**, and **ALTGROUP** commands.

To allow an administrator to assign a non-unique UID or GID using the SHARED keyword, you must grant that administrator at least READ access to the SHARED.IDS profile, as follows:

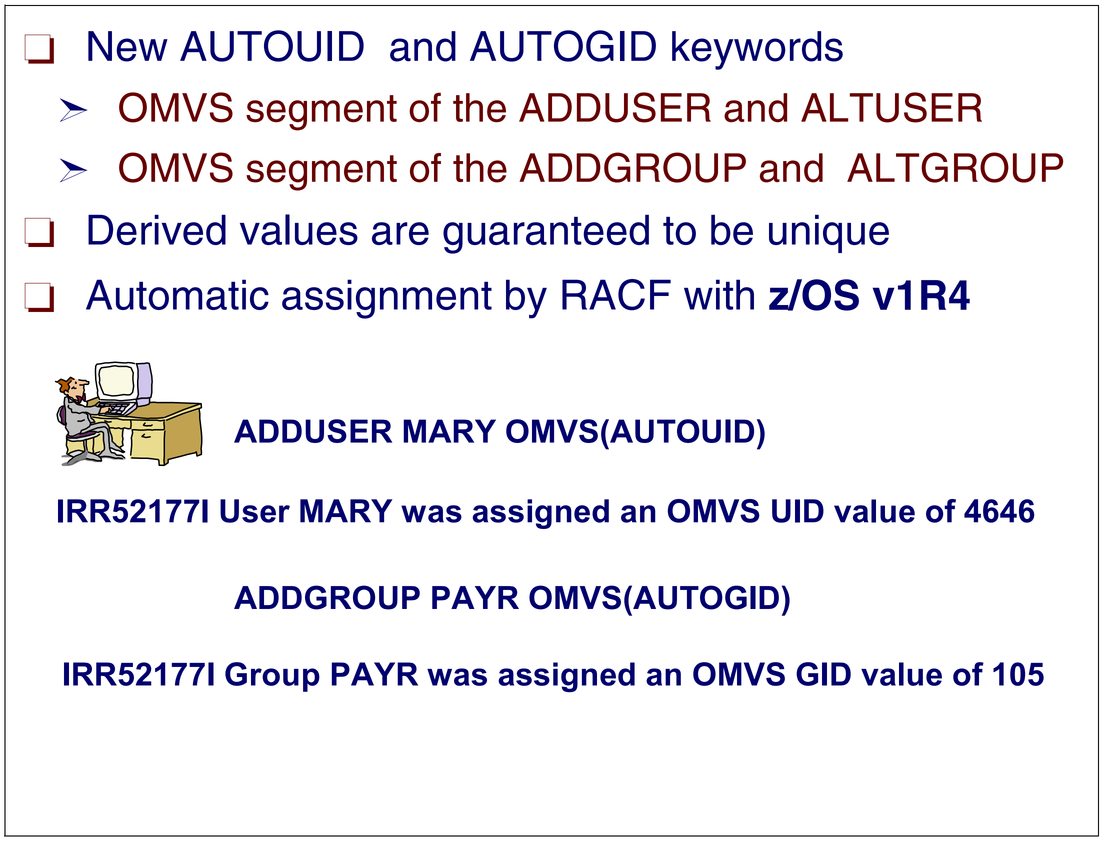
```
PERMIT SHARED.IDS CLASS(UNIXPRIV) ID(UNIXGUY) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Once user ID UNIXGUY has at least READ access to the SHARED.IDS profile, UNIXGUY will be able to assign the same UID or GID to multiple users, using the SHARED KEYWORD, as follows:

```
ADDUSER OMVSKERN OMVS(UID(0) SHARED)
```

Note: To specify the SHARED operand, you must have the SPECIAL attribute or at least READ authority to the SHARED.IDS profile in the UNIXPRIV class.

6.16 Automatic UID and GID assignment



- ❑ New AUTOUID and AUTOGID keywords
  - ➢ OMVS segment of the ADDUSER and ALTUSER
  - ➢ OMVS segment of the ADDGROUP and ALTGROUP
- ❑ Derived values are guaranteed to be unique
- ❑ Automatic assignment by RACF with **z/OS v1R4**

ADDUSER MARY OMVS(AUTOUID)

IRR52177I User MARY was assigned an OMVS UID value of 4646

ADDGROUP PAYR OMVS(AUTOGID)

IRR52177I Group PAYR was assigned an OMVS GID value of 105

Figure 6-16 Assigning UIDs and GIDs automatically

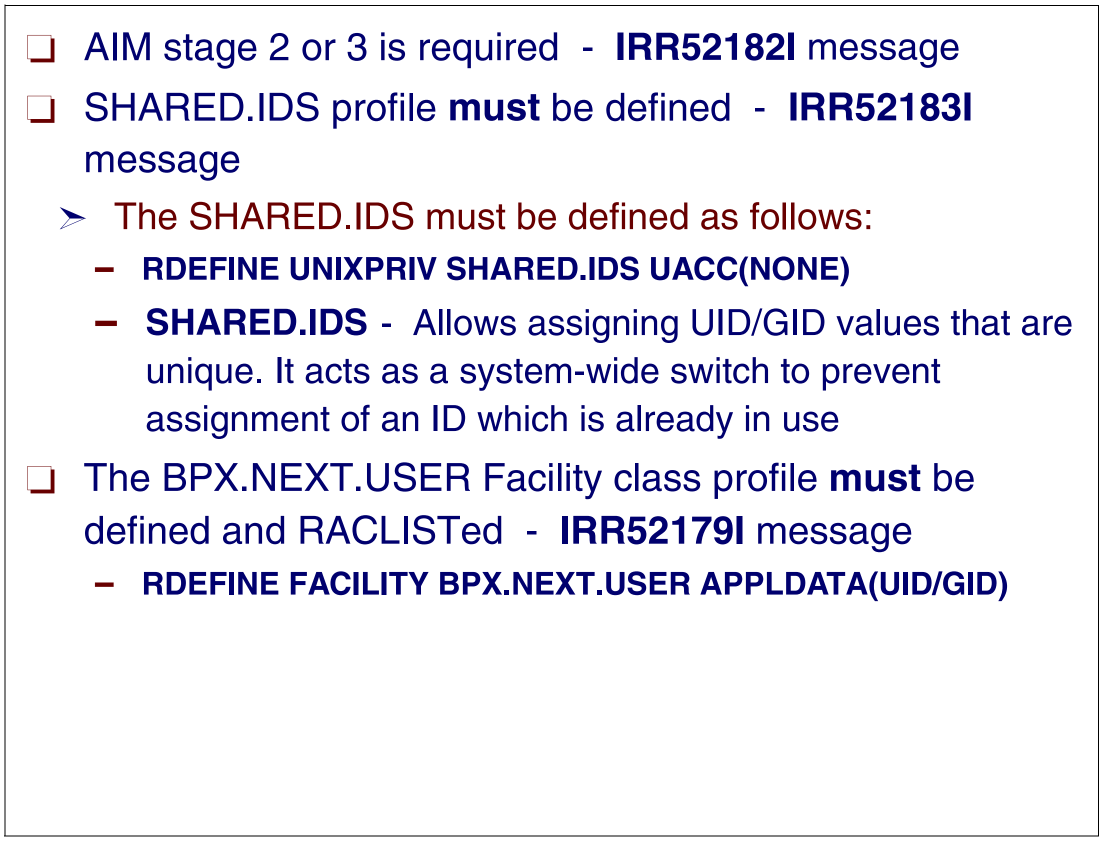
Note: Beginning with z/OS V1R4, you can assign UIDs using the `AUTOUID` and `AUTOGID` keywords

Automatic assignment of UIDs and GIDs

UIDs and GIDs can be assigned automatically by RACF to new users, making it easier to manage the process of assigning UIDs and GIDs to users. Previously, this was a manual process and guaranteed the uniqueness of the UID and GID for every user.

Using a new `AUTOUID` keyword with the **`ADDUSER`** and **`ATLUSER`** commands, an unused UID will be assigned to the new or modified user. Using the `AUTOGID` keyword on **`ADDGROUP`** and **`ALTGROUP`** commands, a GID will be automatically assigned to the new or modified group.

6.17 Automatic assignment requirements



- AIM stage 2 or 3 is required - **IRR52182I** message
- SHARED.IDS profile **must** be defined - **IRR52183I** message
  - The SHARED.IDS must be defined as follows:
    - **RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)**
    - **SHARED.IDS** - Allows assigning UID/GID values that are unique. It acts as a system-wide switch to prevent assignment of an ID which is already in use
- The BPX.NEXT.USER Facility class profile **must** be defined and RACLISTed - **IRR52179I** message
  - **RDEFINE FACILITY BPX.NEXT.USER APPLDATA(UID/GID)**

Figure 6-17 Requirements for automatic assignment of UIDs and GIDs

The use of automatic UID/GID requires the following:

- To use a new RACF profile, SHARED.IDS, the RACF database must have application identity mapping (AIM) stage 2 or 3 implemented. You can convert your RACF database to stage 3 of application identity mapping using the IRRIRA00 conversion utility. See the *z/OS Security Server RACF System Programmer's Guide*, SA22-7681 for information about running the IRRIRA00 conversion utility.

  The IRRIRA00 utility was new in OS/390 V2R10. It converts an existing RACF database to application identity mapping functionality using a four-stage approach.

  Install AIM stage 2 or 3, otherwise, an IRR52182I message is issued and the automatic assignment attempt fails with the following message:

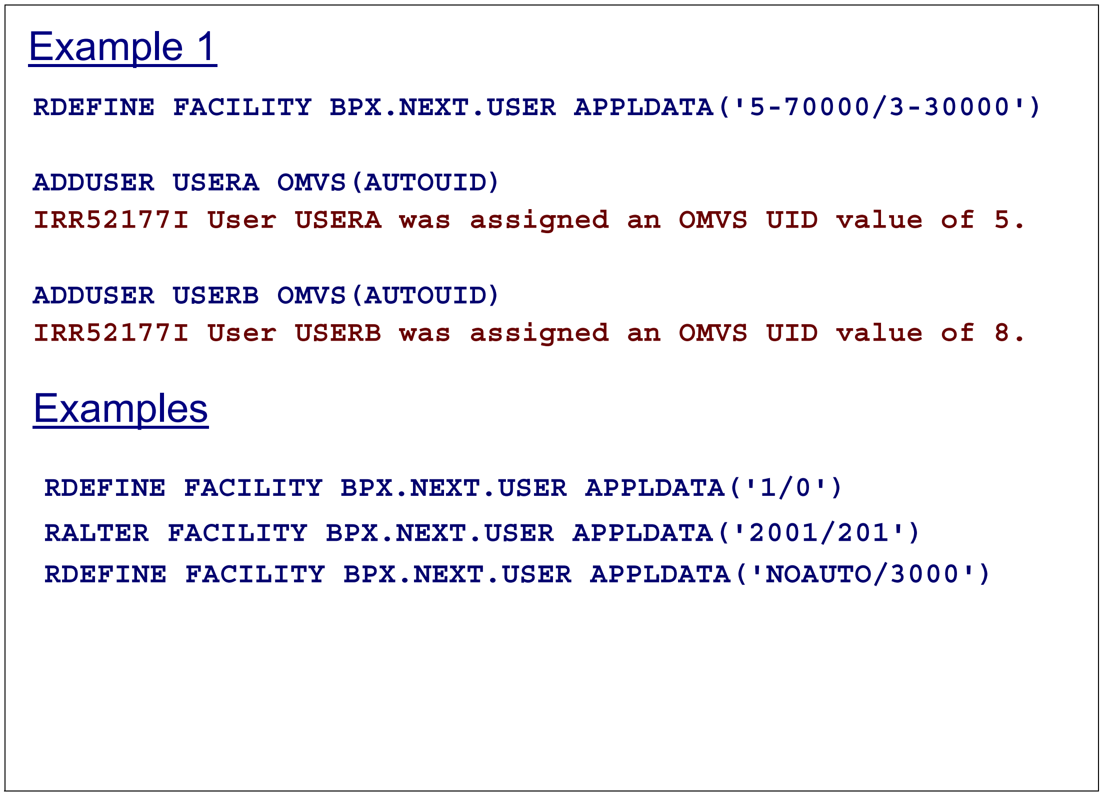
  ```
  IRR52182I Automatic UID assignment requires application identity mapping to be
  implemented.
  ```
- A SHARED.IDS profile must be defined, otherwise, an IRR52183I message is issued and the attempt fails with the following message:

  ```
  IRR52183I Use of automatic UID assignment requires SHARED.IDS to be implemented.
  ```
- The SHARED.IDS must be defined as follows:

  ```
  RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
  ```
- The BPX.NEXT.USER facility class profile must be defined and RACLISTed. Otherwise, an IRR52179I message will be issued and the attempt fails with the following message:

  ```
  IRR52179I The BPX.NEXT.USER profile must be defined before you can use automatic UID
  assignment.
  ```

6.18 Automatic assignment examples



Example 1

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('5-70000/3-30000')

ADDUSER USERA OMVS(AUTOUID)
IRR52177I User USERA was assigned an OMVS UID value of 5.

ADDUSER USERB OMVS(AUTOUID)
IRR52177I User USERB was assigned an OMVS UID value of 8.
```

Examples

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('1/0')
RALTER FACILITY BPX.NEXT.USER APPLDATA('2001/201')
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('NOAUTO/3000')
```

Figure 6-18 Examples of assigning UIDs and GIDs automatically

APPLDATA consists of 2 qualifiers separated by a forward slash (/). The qualifier on the left of the slash character specifies the starting UID value or range of UID values. The qualifier on the right of the slash character specifies the starting GID value or range of GID values. Qualifiers can be null or specified as `'NOAUTO'` to prevent automatic assignment of UIDs or GIDs.

The starting value is the value RACF attempts to use in ID assignment, after determining that the ID is not in use. If it is in use, the value is incremented until an appropriate value is found.

The maximum value valid in the APPLDATA specification is 2,147,483,647. If this value is reached or a candidate UID/GID value has been exhausted for the specified range, subsequent automatic ID assignment attempts fail and message IRR52181I is issued.

In the following example, we have defined the APPLDATA for a range of values from 5 to 70000 for UIDs and from 3 to 30000 for GIDs. USERA and USERB are added using the automatic assignment of UID. The range of automatic UID assignment starts with 5, so USERA is assigned to UID(5) which was free. UID(6) and UID(7) were already assigned before we started our example, so the next free UID is 8. USERB is assigned to UID(8).

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('5-70000/3-30000')

ADDUSER USERA OMVS(AUTOUID)
IRR52177I User USERA was assigned an OMVS UID value of 5.

ADDUSER USERB OMVS(AUTOUID)
IRR52177I User USERB was assigned an OMVS UID value of 8.
```

RACF extracts the APPLDATA from the BPX.NEXT.USER and parses out the starting value. It checks if it is already in use and if so, the value is incremented and checked again until an unused value is found. Once a free value is found, it assigns the value to the user or group and replaces the APPLDATA with the new starting value, which is the next potential value or the end of the range.

In our example, that means that if UID(6) becomes free after UID(7) is assigned to USERB, RACF will start checking from UID(8) in the next assignment, so it will not assign UID(6). But you can change the APPLDATA and modify the starting value. The APPLDATA can be changed using the following command:

```
RALTER FACILITY BPX.NEXT.USER APPLDATA('2000/500')
```

Note: Automatic assignment of UIDs and GIDs fails if you specify a list of users to be defined with the same name or if you specify the `SHARED` keyword. Also, `AUTOUID` or `AUTOGID` is ignored if UID or GID is also specified.

APPLDATA examples

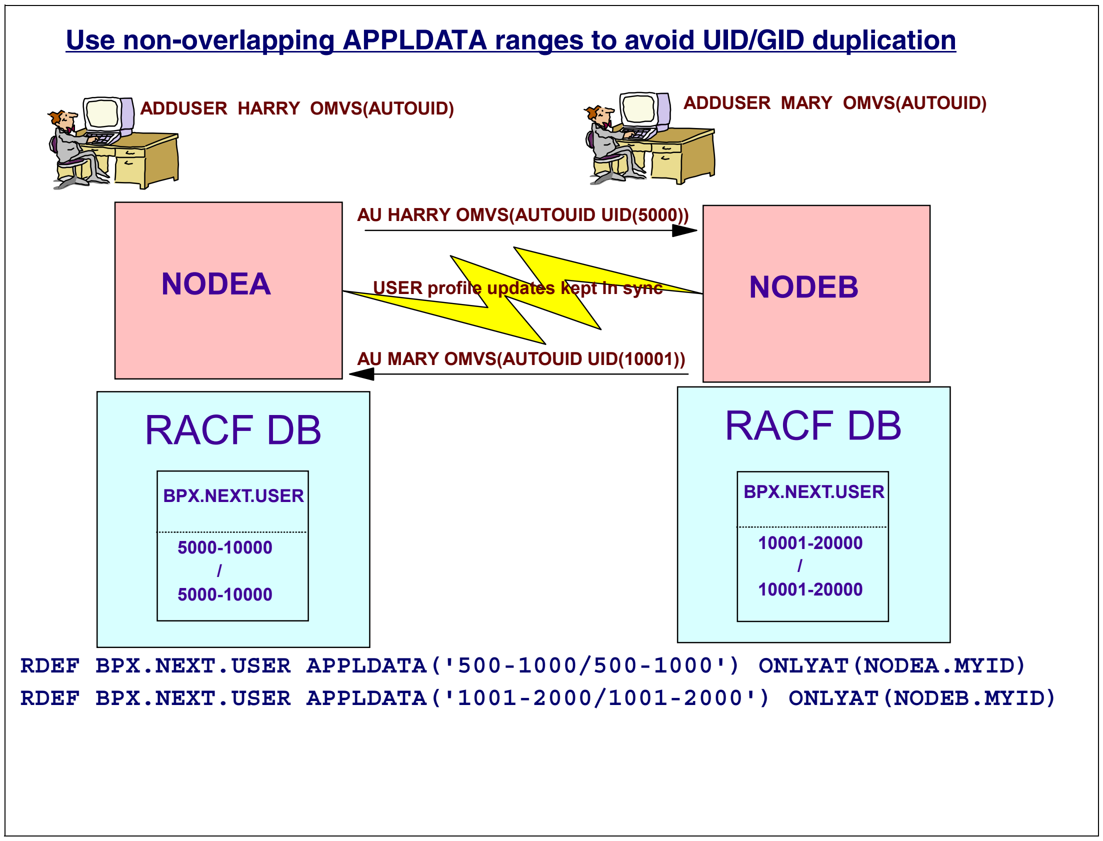
Here are some examples of correct and incorrect APPLDATA specifications:

Good data	1/0 1-50000/1-20000 NOAUTO/100000 /100000 10000-20000/NOAUTO 10000-20000/
Bad data	123B / 2147483648 /* higher than max UID value */ 555/1000-900

If you have an incorrect specification and attempt to use `AUTOUID` on an **`ADDUSER`** command, the following message is issued:

```
IRR52187I Incorrect APPLDATA syntax for the BPX.NEXT.USER profile.
```

6.19 Automatic assignment with RRSF


Use non-overlapping APPLDATA ranges to avoid UID/GID duplication
ADDUSER HARRY OMVS(AUTOUID)
ADDUSER MARY OMVS(AUTOUID)
AU HARRY OMVS(AUTOUID UID(5000))
NODEA
USER profile updates kept in sync
NODEB
AU MARY OMVS(AUTOUID UID(10001))
RACF DB
BPX.NEXT.USER
5000-10000
/
5000-10000
RACF DB
BPX.NEXT.USER
10001-20000
/
10001-20000

```
RDEF BPX.NEXT.USER APPLDATA('500-1000/500-1000') ONLYAT(NODEA.MYID)
RDEF BPX.NEXT.USER APPLDATA('1001-2000/1001-2000') ONLYAT(NODEB.MYID)
```

Figure 6-19 Automatic assignment with RRSF

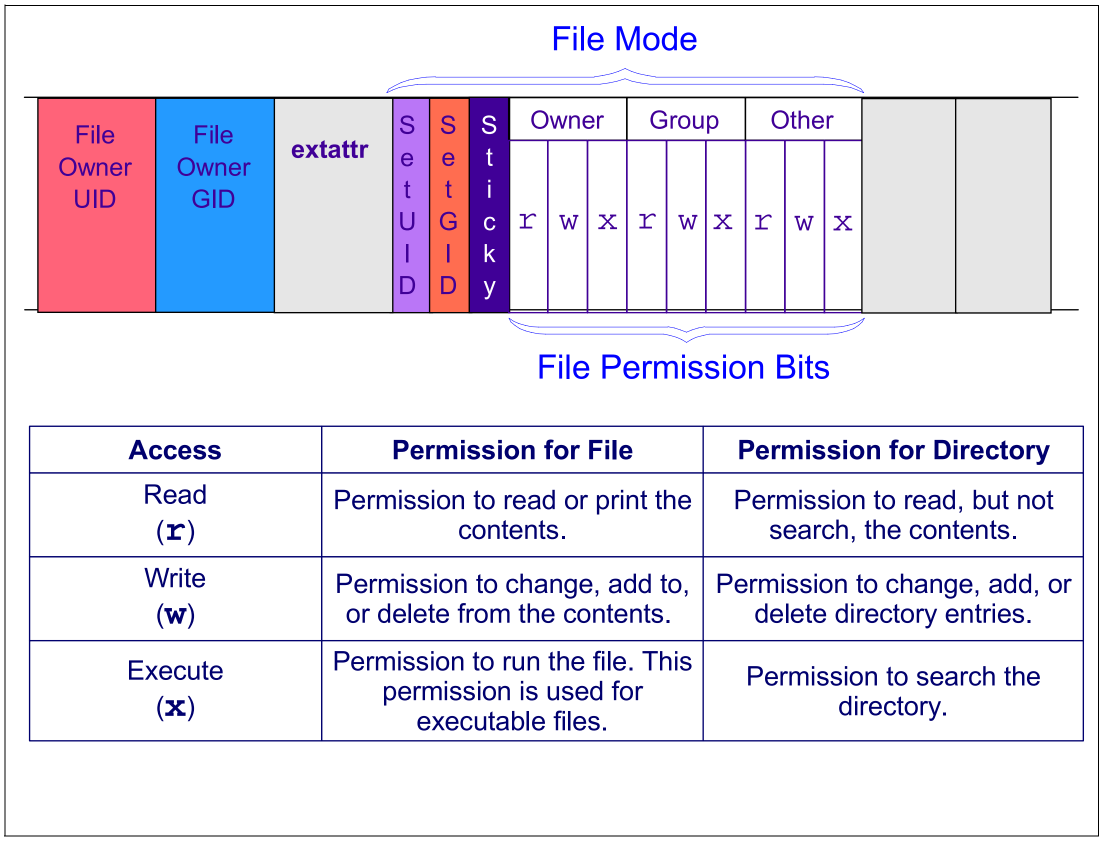
In an RRSF configuration, shown in Figure 6-19, use non-overlapping APPLDATA ranges to avoid UID and GID duplications.

An additional concern is to make RACF automatically suppress propagation of internal updates. This can be done by specifying the `ONLYAT` keyword to manage the BPX.NEXT.USER profile, as follows:

```
RDEFINE BPX.NEXT.USER APPLDATA('5000-10000/5000-10000') ONLYAT(NODEA.MYID)
RDEFINE BPX.NEXT.USER APPLDATA('10001-20000/10001-20000') ONLYAT(NODEB.MYID)
```

For more information about automatic assignment in a RRSF configuration, refer to *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683.

6.20 z/OS UNIX security: File security packet



Access	Permission for File	Permission for Directory
Read (**r**)	Permission to read or print the contents.	Permission to read, but not search, the contents.
Write (**w**)	Permission to change, add to, or delete from the contents.	Permission to change, add, or delete directory entries.
Execute (**x**)	Permission to run the file. This permission is used for executable files.	Permission to search the directory.

Figure 6-20 File security packet definitions

Each z/OS UNIX file and directory has a file security packet (FSP) associated with it to control access. The FSP is created when a file or directory is created. It is stored in the file system for the life of the file or directory, until the file or directory is deleted, at which time the FSP is also deleted.

The FSP consists of:

- File owner UID
- File owner GID
- File mode

File Mode

The file mode consists of:

SetUID This bit only relates to executable files. If on, it causes the UID of the user executing the file to be set to the file's UID.

SetGID This bit only relates to executable files. If on, it causes the GID of the user executing the file to be set to the file's GID.

Sticky Bit This bit only relates to executable files. If on, it causes the file to be retained in memory for performance reasons. The implementation of this varies between platforms. In z/OS UNIX, it means programs are loaded from LPA (or LNKLST as per normal MVS program search) instead of an HFS file. For a directory, the sticky bit causes UNIX to permit files in a directory or subdirectories to be deleted or renamed only by the owner of the file, or by the owner of the directory, or by a superuser.

Another section of the FSP, which is specific to the z/OS UNIX implementation, is called Extended Attributes (extattr). It contains flags to mark HFS program files as APF-authorized and program controlled. A shell command called **`extattr`** is used to manipulate these bits.

The file mode also has the file permission bits, consisting of:

- Owner read/write/execute permissions
- Group read/write/execute permissions
- Other (or all users) read/write/execute permissions

where:

r Read (`r`) access to both files and directories

w Write (`w`) access to both files and directories

x Execute (`x`) has a different meaning for files and directories, as follows:

- For an executable file, an access of `x` means that the user can execute the file.

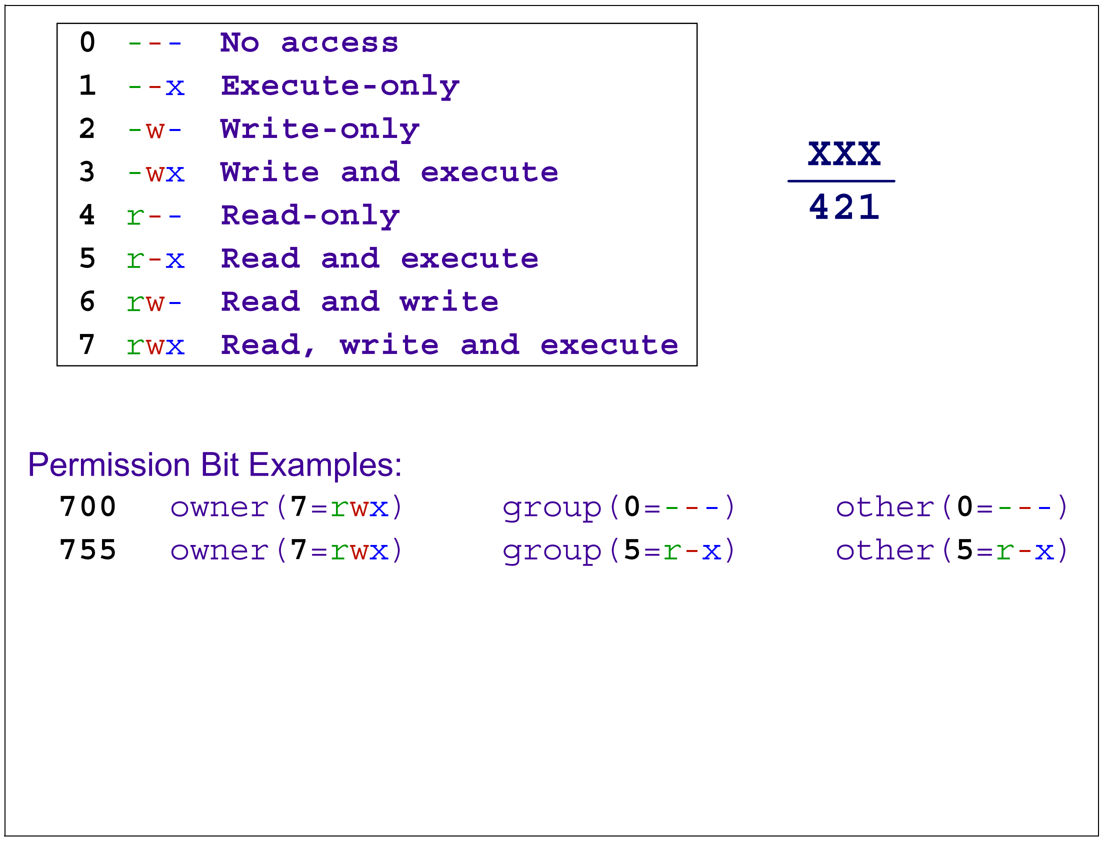
- For a directory, an access of `x` means the user can search the directory.

Both read (`r`) and execute (`x`) are required in order to execute a shell script. To access HFS files, a user needs the following:

- Search (`x`) permission to all the directories in the pathname of files the user wants to access.
- Write permission to directories where the user will be creating new files and directories.
- Read and/or write permission, as appropriate, to files for access.
- Execute (`x`) permission for an executable file.

Note: In z/OS UNIX, these three permissions are not hierarchical. For example, a user with write permission who does *not* have read permission, can only write over existing data or add data to a file, and cannot look at the contents of the file or print the file. Similarly, write and read permission does not allow a user to execute a file or search a directory.

6.21 Octal values for permission bits



0	---	No access
1	--x	Execute-only
2	-w-	Write-only
3	-wx	Write and execute
4	r--	Read-only
5	r-x	Read and execute
6	rw-	Read and write
7	rwx	Read, write and execute

$$\frac{\text{XXX}}{421}$$

Permission Bit Examples:

700	owner(7=rwx)	group(0=---)	other(0=---)
755	owner(7=rwx)	group(5=r-x)	other(5=r-x)

Figure 6-21 Octal values of permission bit settings

z/OS UNIX commands typically allow the definition of permission bit settings using octal notation. It is a simpler way to describe the permission bit string.

The octal numbers relate to the bit positions as follows:

```
Dec    Bin     Map
===    ===     ===
0   =  000  =  ---
1   =  001  =  --x
2   =  010  =  -w-
3   =  011  =  -wx
4   =  100  =  r--
5   =  101  =  r-x
6   =  110  =  rw-
7   =  111  =  rwx
```

where:

Dec	Decimal representation of the octal value
Bin	Binary representation of the octal value
Map	Map of permission bits according to binary positions

UNIX commands such as **`chmod`** accept octal notation when referring to permission bit settings.

A permission bit setting of 700 is good for a user's private files, allowing the owner full access while denying any access to others.

A permission bit setting of 755 is good for a user to let other people access their files (read and execute), but not update them.

6.22 Data set security versus file security

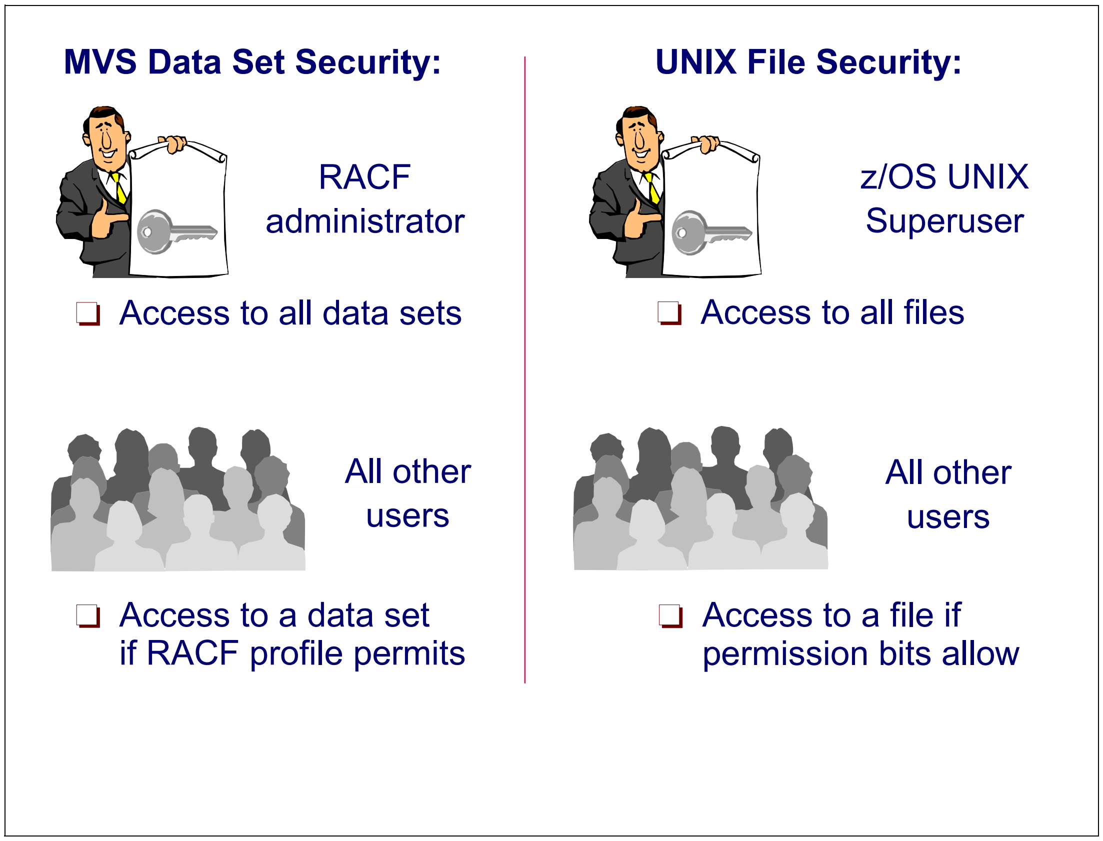


Figure 6-22 MVS data set security versus file system security

Security processing within z/OS UNIX differs in many ways from standard security processing in MVS. MVS resources like users and data are protected by RACF profiles stored in the RACF database. RACF refers to the profiles when deciding which users should be permitted to protected system resources. Security administration is done with RACF commands or RACF ISPF panels.

z/OS UNIX users are defined as MVS users and they are administrated by RACF profiles. The security information for files and directories in a hierarchical file system is stored within the file system itself in a file security packet (FSP). HFS files and directories are protected by permission bit information which is kept in the FSP. Administration of file security is performed by using z/OS UNIX shell commands, or ISHELL menu options.

The user administration is similar for regular MVS users and z/OS UNIX users. Every user must present a password when logging on to the system. z/OS UNIX uses a UID and GID for each user and this information is stored in RACF profiles together with the user ID and password information. The concept of a superuser in z/OS UNIX is similar to a RACF security administrator.

z/OS UNIX users do not work with data sets, they work with files and directories. z/OS UNIX users do not have to be aware that their data is located physically in an HFS data set. All they see is the hierarchical file structure made up of multiple mounted HFS data sets. The FSPs are maintained by z/OS UNIX commands. RACF data set profiles cannot be used to protect z/OS UNIX files and directories.

6.23 z/OS UNIX user's security environment

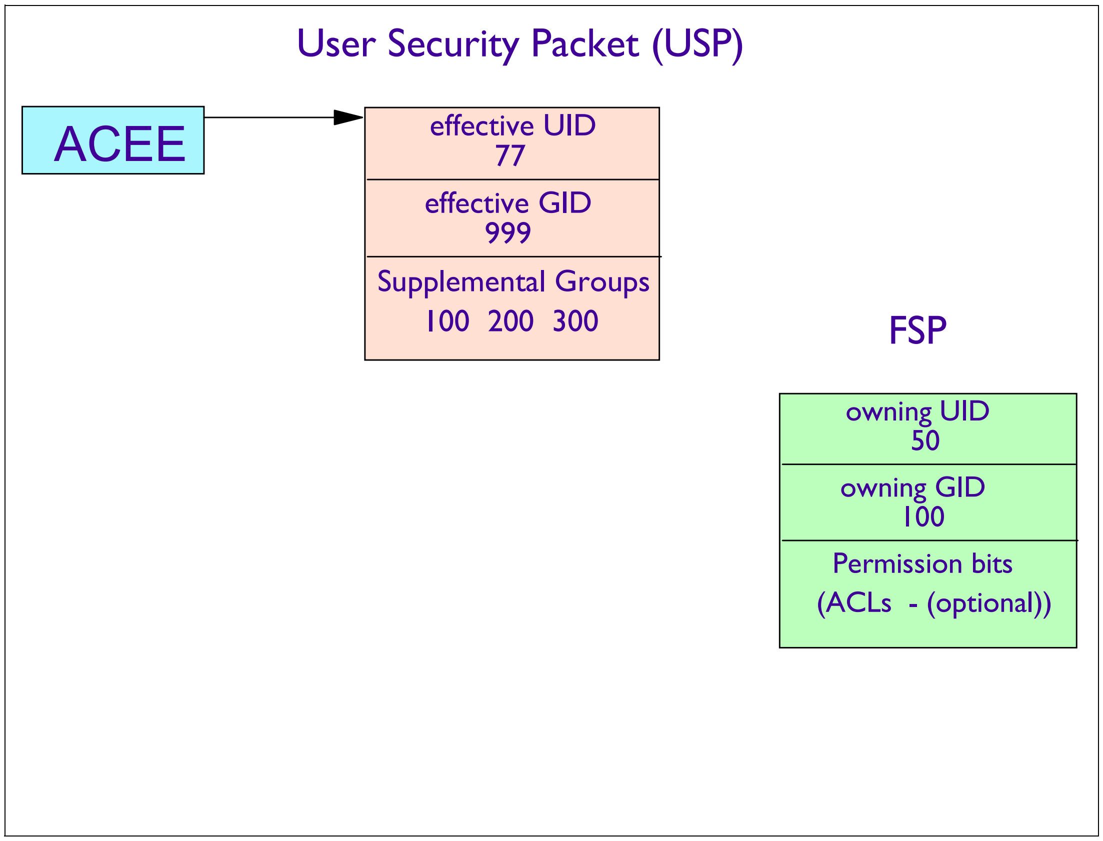


Figure 6-23 A z/OS UNIX user's file security environment

Authorization checking for z/OS UNIX files and directories uses the control blocks shown in Figure 6-23, and RACF makes the following checks:

- The accessor environment element (ACEE) is 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.
- The effective UID and effective GID of the process are used in determining access decisions. The only exception is that if file access is being tested, rather than requested, the real UID and GID are used instead of the effective UID and GID. The real and effective IDs are generally the same for a process, but if a set-uid or set-gid program is executed, they can be different.

6.24 Access checking flows

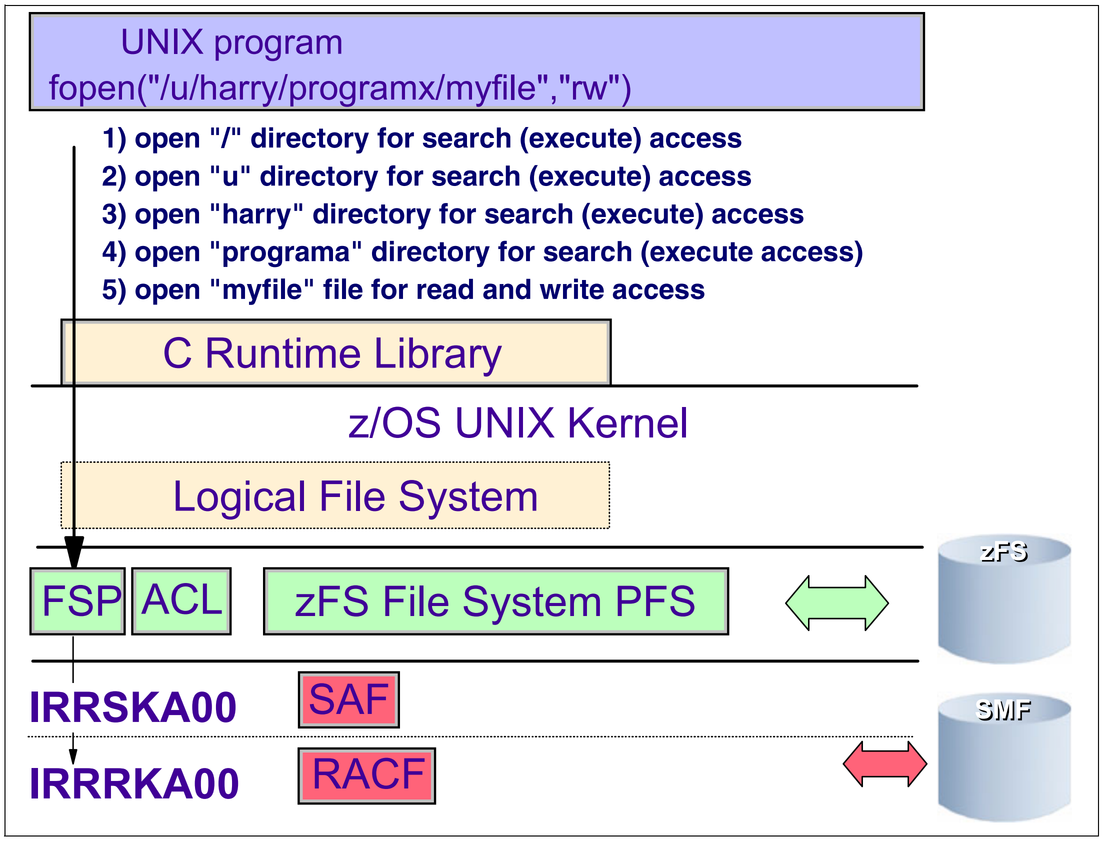

UNIX program
fopen("/u/harry/programx/myfile","rw")
1) open "/" directory for search (execute) access
2) open "u" directory for search (execute) access
3) open "harry" directory for search (execute) access
4) open "programa" directory for search (execute access)
5) open "myfile" file for read and write access
C Runtime Library
z/OS UNIX Kernel
Logical File System
FSP
ACL
zFS File System PFS
zFS
IRRSKA00
SAF
SMF
IRRRKA00
RACF

Figure 6-24 Access checking flow for access to a file

Figure 6-24 shows the access checking flow from a UNIX program, or the access could be from a z/OS UNIX shell user, to the security product. The z/OS UNIX kernel calls the file system, and the file system calls the security product. The kernel calls the file system iteratively for each directory component of the path name (one is required in order to locate the next). Then, the base file name is retrieved. For each directory lookup, the file system calls the security product to make sure the user has search authority. Then the security product is called to insure the user has the requested access to the base file.

This is the basic architecture of every UNIX file system.

6.25 File authorization checking flow

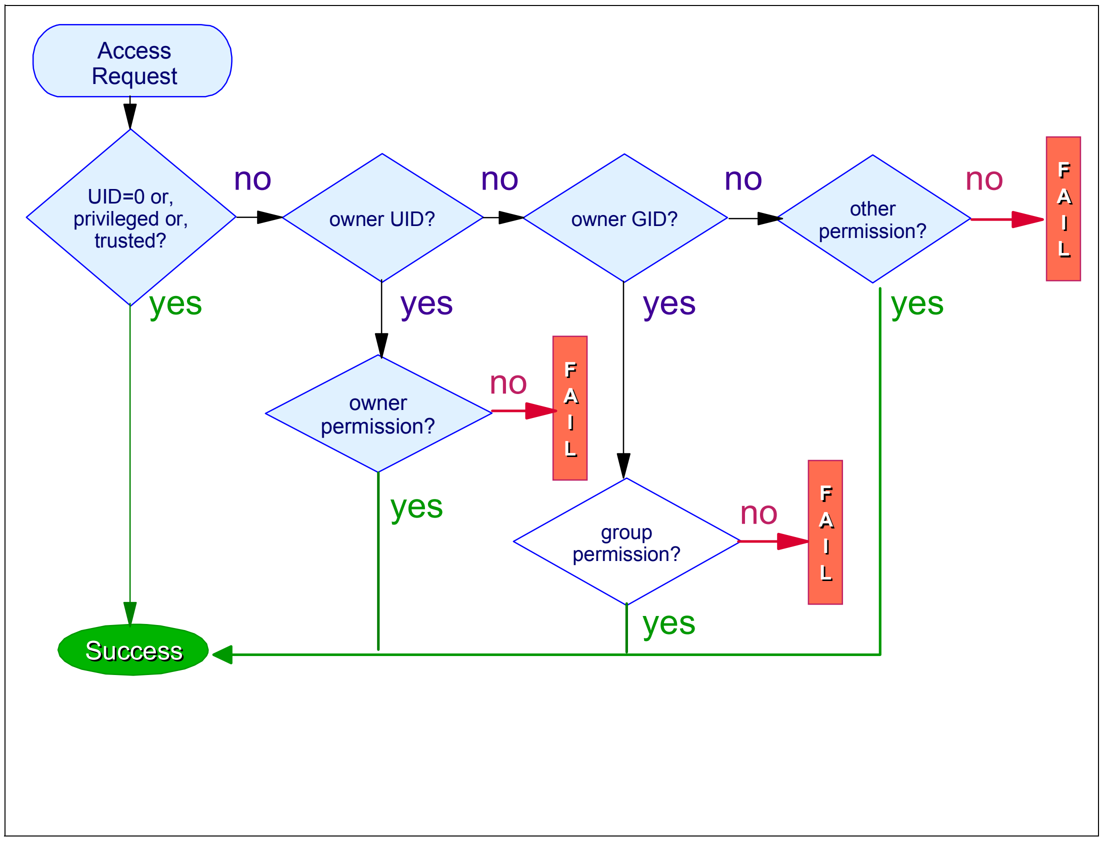


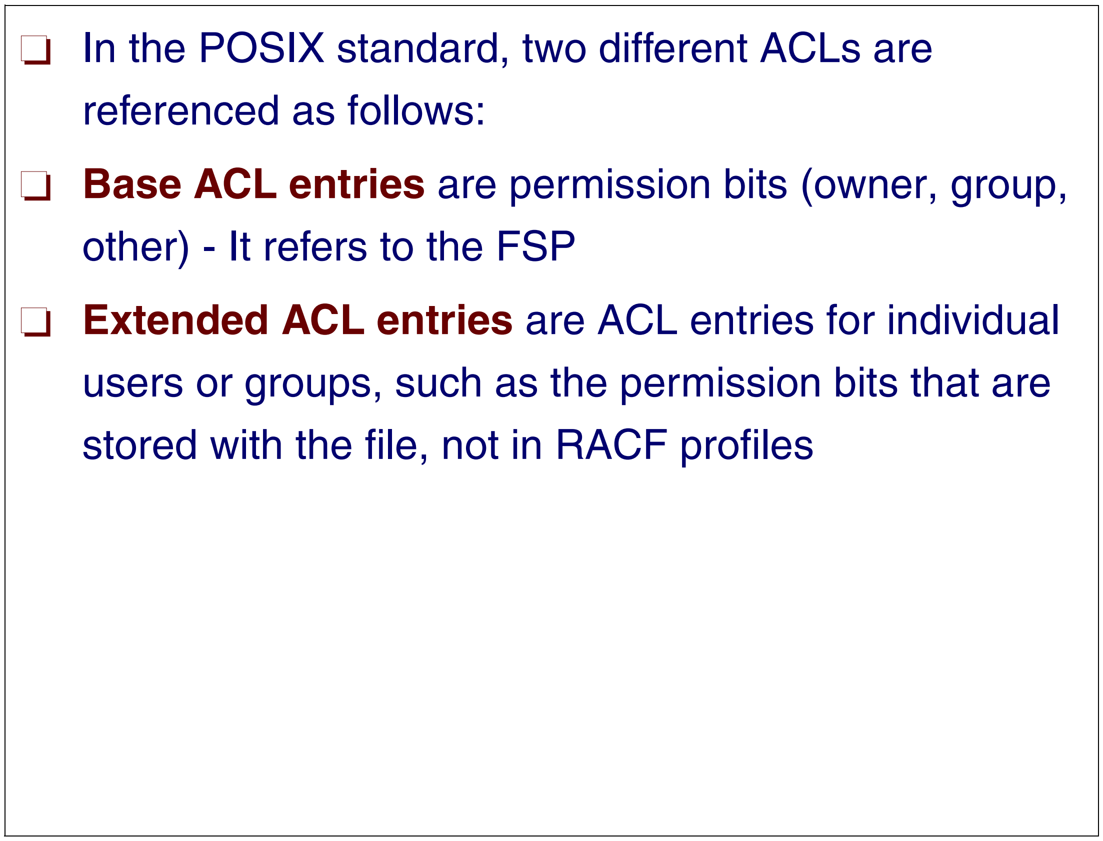
Figure 6-25 Authorization checking with z/OS UNIX

Authorization checking is done for all directories and files (including special files) in the file system. z/OS UNIX calls RACF to perform the authorization checking and passes RACF the FSP (file security packet), and the CRED (security credentials).

Figure 6-25 shows the sequence of authorization checks, as follows:

- A superuser (UID of zero) is allowed access to all resources.
- If the effective UID of the process (the accessor) equals the UID of the file, RACF uses the owner permissions in the FSP to either allow or deny access.
- If the effective GID of the process equals the GID of the file, RACF uses the group permissions in the FSP to either allow or deny access. If RACF list-of-groups checking is active (SETROPTS GRPLIST), RACF will look at the user's connect groups that have a GID for a group that matches the GID of the file. If it finds a matching GID, RACF will allow or deny access based on the group permissions specified in the FSP. Note that if a user is connected to more that 300 z/OS UNIX groups, only the first 300 will be used.
- If the effective UID or GID of the process does not match the file UID or GID, then the other permission bits will determine access.

6.26 POSIX standard and UNIX ACLs



- ❑ In the POSIX standard, two different ACLs are referenced as follows:
- ❑ **Base ACL entries** are permission bits (owner, group, other) - It refers to the FSP
- ❑ **Extended ACL entries** are ACL entries for individual users or groups, such as the permission bits that are stored with the file, not in RACF profiles

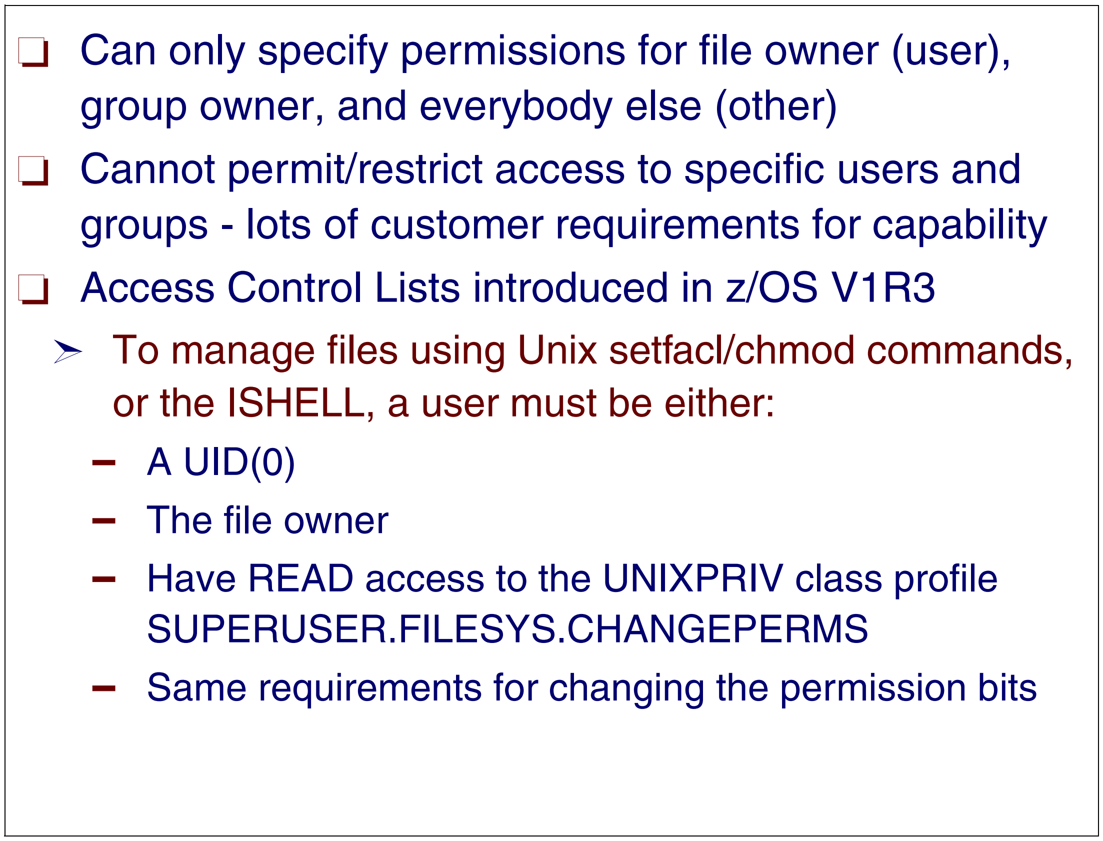
Figure 6-26 The POSIX standard for ACL support

ACLs have existed on various UNIX platforms for many years, but with variations in the interfaces. ACL support in z/OS V1R3 is based on a POSIX standard that was never implemented when z/OS UNIX was first introduced as OpenEdition.

In the POSIX standard, two different ACLs are referenced as follows:

- Base ACL entries are permission bits (owner, group, other). This refers to the FSP. 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, such as the permission bits that are stored with the file, not in RACF profiles. Extended ACL entries 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.

6.27 Limitations of current permission bits



- ❑ Can only specify permissions for file owner (user), group owner, and everybody else (other)
- ❑ Cannot permit/restrict access to specific users and groups - lots of customer requirements for capability
- ❑ Access Control Lists introduced in z/OS V1R3
  - ➢ To manage files using Unix setfacl/chmod commands, or the ISHELL, a user must be either:
    - – A UID(0)
    - – The file owner
    - – Have READ access to the UNIXPRIV class profile SUPERUSER.FILESYS.CHANGEPERMS
    - – Same requirements for changing the permission bits

Figure 6-27 z/OS UNIX limitation with POSIX standard on ACLs

Access control lists (ACLs) are introduced in z/OS V1R3 as a way to provide a greater granularity for access to z/OS UNIX files and directories. It is based on a POSIX standard that was never approved, and other UNIX implementations.

z/OS V1R3 provides support for ACLs to control access to files and directories for:

- ► Specific individual user or users (UID)
- ► Specific group or groups (GID)

To create an ACL for a file, you must have one of the following security access controls:

- ► Be the file owner
- ► BPX.SUPERUSER
- ► Have superuser authority (UID=0)
- ► Have READ access to the SUPERUSER.FILESYS.CHANGEPERMS profile in the UNIXPRIV class

6.28 FSPs and ACLs

- There are 3 types of ACLs
  - Access ACLs
  - File default model ACLs
  - Directory default model ACLs

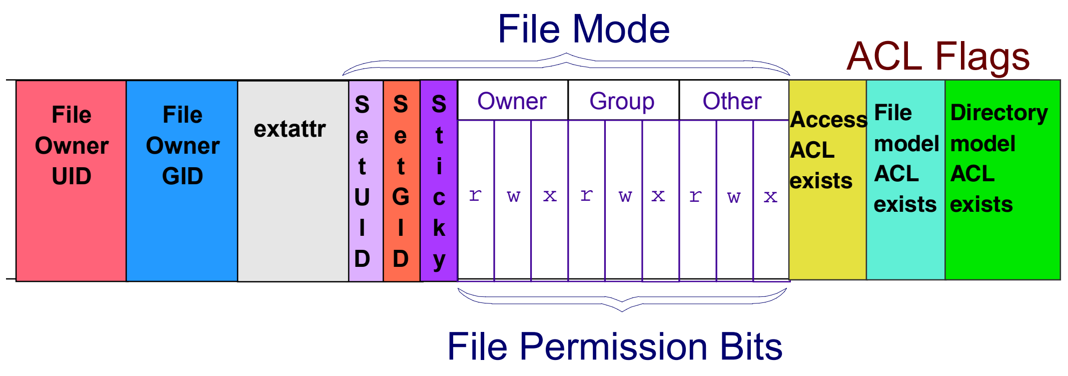


Figure 6-28 File security packet and ACL support

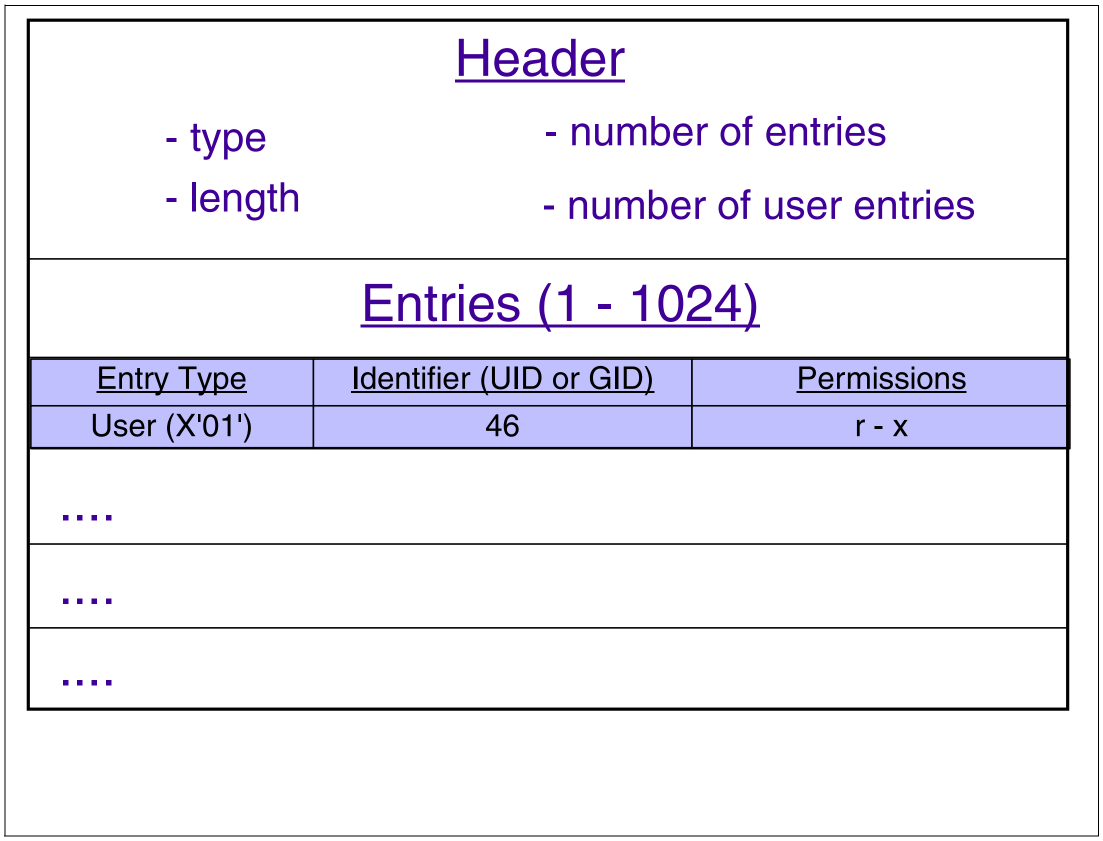
ACL types

To reduce administrative overhead, three types of ACLs (extended ACLs) are defined, giving the capability to inherit ACLs to newly created files and directories as follows:

Access ACLs	This type of ACL is used to provide protection for a file system object (specific for a file or directory).
File default ACLs	This type is a model ACL that is inherited by files created within the parent directory. 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.
Directory default ACLs	This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL.

ACLs are used together with the permission bits in the FSP to control the access to z/OS UNIX files and directories by individual users (UIDs) and groups (GIDs).

6.29 Access control list table



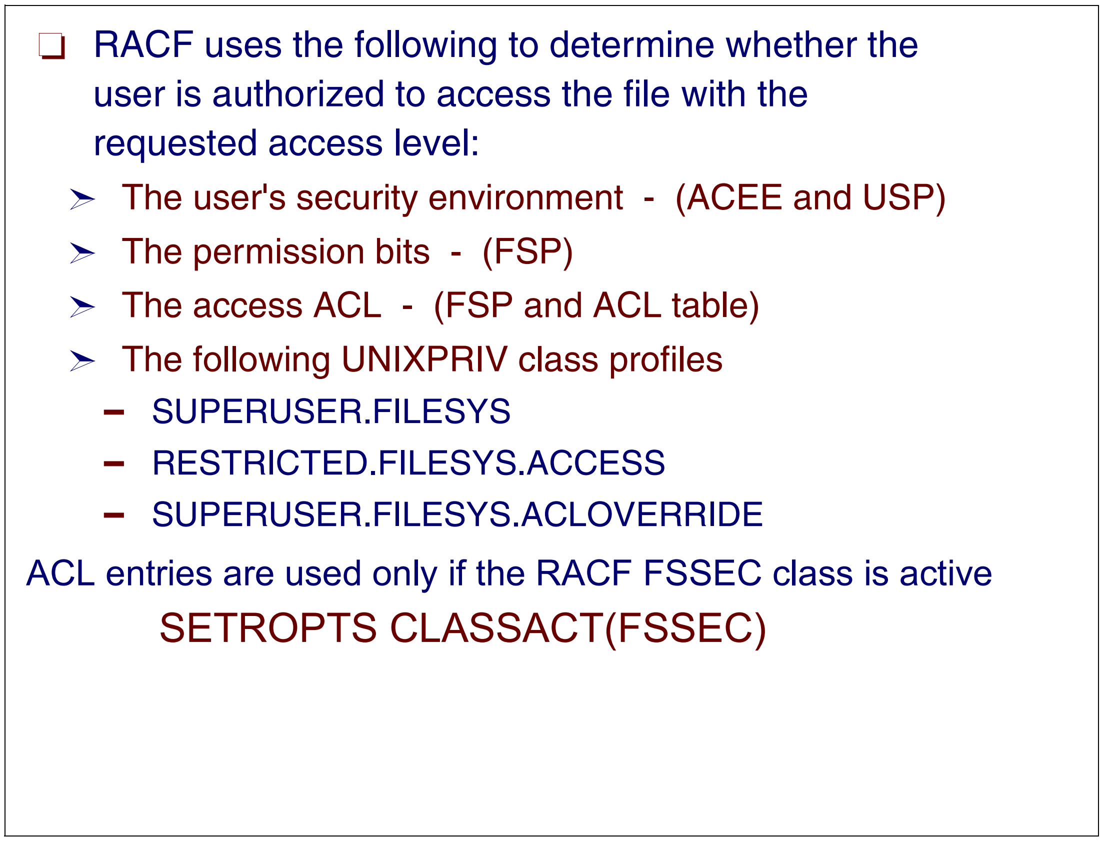
Entry Type	Identifier (UID or GID)	Permissions
User (X'01')	46	r - x

Figure 6-29 ACL table that contains ACL entries after creation

An ACL is mapped by the SAF IRRPFACL macro, as shown inFigure 6-29, where the set of user entries is followed by the set of group entries.

The entries are sorted in ascending order by UID and GID to help optimize the access checking algorithm. The table consists of a list of entries (with a maximum of 1024) where every entry has information about the type (user or group), identifier (UID or GID), and permissions (read, write, and execute) to apply to a file or directory.

6.30 File authorization check summary



- RACF uses the following to determine whether the user is authorized to access the file with the requested access level:
  - The user's security environment - (ACEE and USP)
  - The permission bits - (FSP)
  - The access ACL - (FSP and ACL table)
  - The following UNIXPRIV class profiles
    - SUPERUSER.FILESYS
    - RESTRICTED.FILESYS.ACCESS
    - SUPERUSER.FILESYS.ACLOVERRIDE

ACL entries are used only if the RACF FSSEC class is active

SETROPTS CLASSACT(FSSEC)

Figure 6-30 Authorization checks made for access to files and directories

Authorization checking for z/OS UNIX files and directories is done by RACF, which makes the following checks:

- The accessor environment element (ACEE) is 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.
- The effective UID and effective GID of the process are used in determining access decisions. The only exception is that if file access is being tested, rather than requested, the real UID and GID are used instead of the effective UID and GID. The real and effective IDs are generally the same for a process, but if a set-uid or set-gid program is executed, they can be different.
- If the GID matches the file owner GID, the file's “group” permission bits are checked. If the group bits allow the requested access, then access is granted.

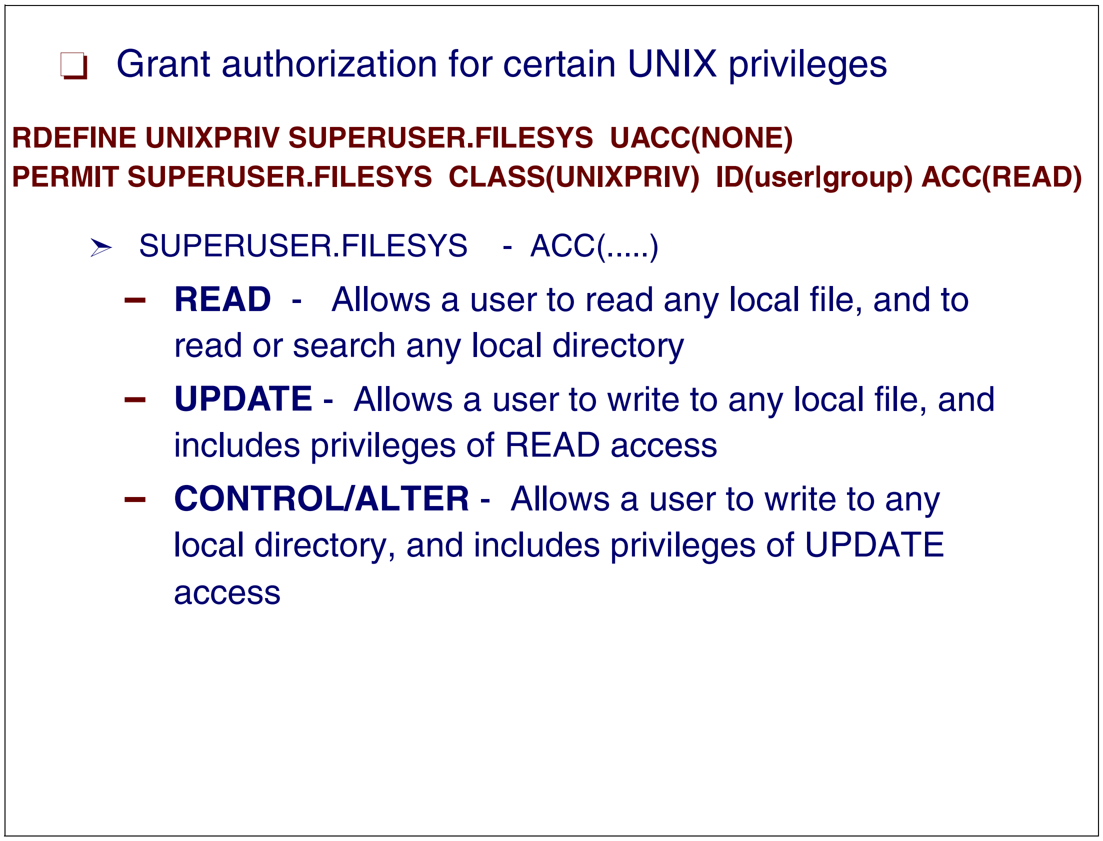
  If any of the user's supplemental GIDs match the file owner GID, the file's group permission bits are checked. If the group bits allow the requested access, then access is granted.

  If no group bits access is allowed and the FSSEC class is active, and an ACL exists, and there is an ACL entry for any of the user's supplemental GIDs, then the permission bits of that ACL entry are checked. If at least one matching ACL entry was found for the GID, or any of the supplemental GIDs, then processing continues with the ACLOVERRIDE checking.

If no group ACL matches, then if the UNIXPRIV class is active, the SUPERUSER.FILESYS access is checked.

- SUPERUSER.FILESYS.ACLOVERRIDE is checked only when a user's access was denied by a matching ACL entry based on the user's UID or one of the user's GIDs. If the user's access was denied by the file's permission bits, SUPERUSER.FILESYS is checked.

6.31 Profile in UNIXPRIV class



- Grant authorization for certain UNIX privileges

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS  UACC(NONE)
PERMIT SUPERUSER.FILESYS  CLASS(UNIXPRIV)  ID(user|group) ACC(READ)
```

- SUPERUSER.FILESYS - ACC(.....)
  - **READ** - Allows a user to read any local file, and to read or search any local directory
  - **UPDATE** - Allows a user to write to any local file, and includes privileges of READ access
  - **CONTROL/ALTER** - Allows a user to write to any local directory, and includes privileges of UPDATE access

Figure 6-31 SUPERUSER.FILESYS profile in UNIXPRIV class

This profile in the UNIXPRIV class was introduced in OS/390 V2R8. The UNIXPRIV class provided the capability to assign specific superuser functions to a user or group when you give a user or group either a:

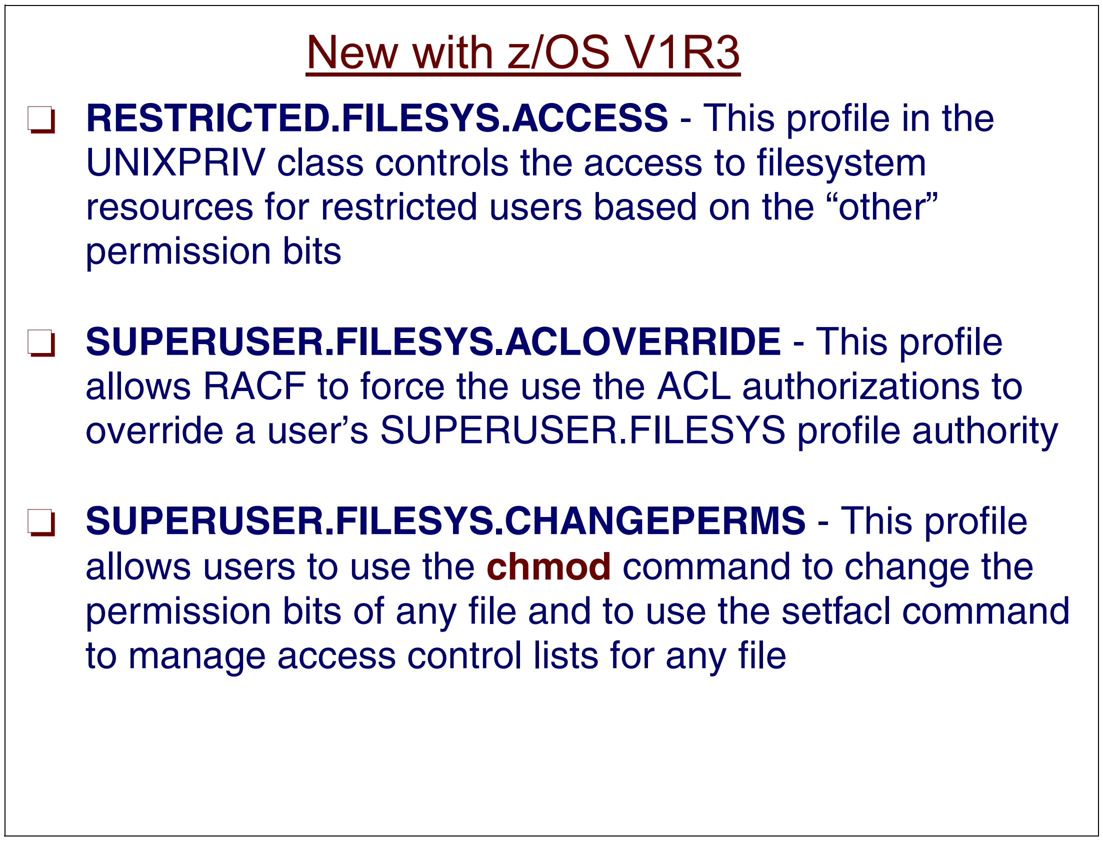
- UID of 0
- BPX.SUPERUSER profile

Either of these gives a user or group access to all UNIX functions and resources. A BPX.SUPERUSER profile allows you to request that you be given such access, but you do not have the access unless you make the request. So, instead of giving a user or group access to all functions a superuser has, the UNIXPRIV class provides profiles that allow access to specific superuser functions.

The SUPERUSER.FILESYS profile in the UNIXPRIV class has three access levels that allow access to z/OS UNIX files as follows:

READ	Allows a user to read any local file, and to read or search any local directory.
UPDATE	Allows a user to write to any local file, and includes privileges of READ access.
CONTROL/ALTER	Allows a user to write to any local directory, and includes privileges of UPDATE access.

6.32 New profiles in UNIXPRIV class



New with z/OS V1R3

- ❑ **RESTRICTED.FILESYS.ACCESS** - This profile in the UNIXPRIV class controls the access to filesystem resources for restricted users based on the “other” permission bits
- ❑ **SUPERUSER.FILESYS.ACLOVERRIDE** - This profile allows RACF to force the use the ACL authorizations to override a user’s SUPERUSER.FILESYS profile authority
- ❑ **SUPERUSER.FILESYS.CHANGEPERMS** - This profile 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

Figure 6-32 UNIXPRIV profiles introduced with z/OS V1R3

RESTRICTED.FILESYS.ACCESS profile

This profile specifies that RESTRICTED users cannot gain file access by virtue of the “other” permission bits. Checking for this new profile RESTRICTED.FILESYS.ACCESS is done for RESTRICTED users regardless of whether an ACL exists, so this function can be exploited whether you plan to use ACLs or not. You can define the profile as follows:

```
RDEFINE UNIXPRIV RESTRICTED.FILESYS.ACCESS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

SUPERUSER.FILESYS.ACLOVERRIDE profile

Any user who is not a superuser with UID(0) or the file owner, and who is denied access through the ACL, can still access a file system resource if the user has sufficient authority to the SUPERUSER.FILESYS resource in the UNIXPRIV class. Therefore, to prevent this, you can force RACF to use your ACL authorizations to override a user's SUPERUSER.FILESYS authority by defining the following profiles:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) RACLIST
```

SUPERUSER.FILESYS.CHANGEPERMS profile

As an enhancement to superuser granularity, when using the **`chmod`** command, a RACF service (IRRSCF00) has been updated to check the caller's authorization to the resource

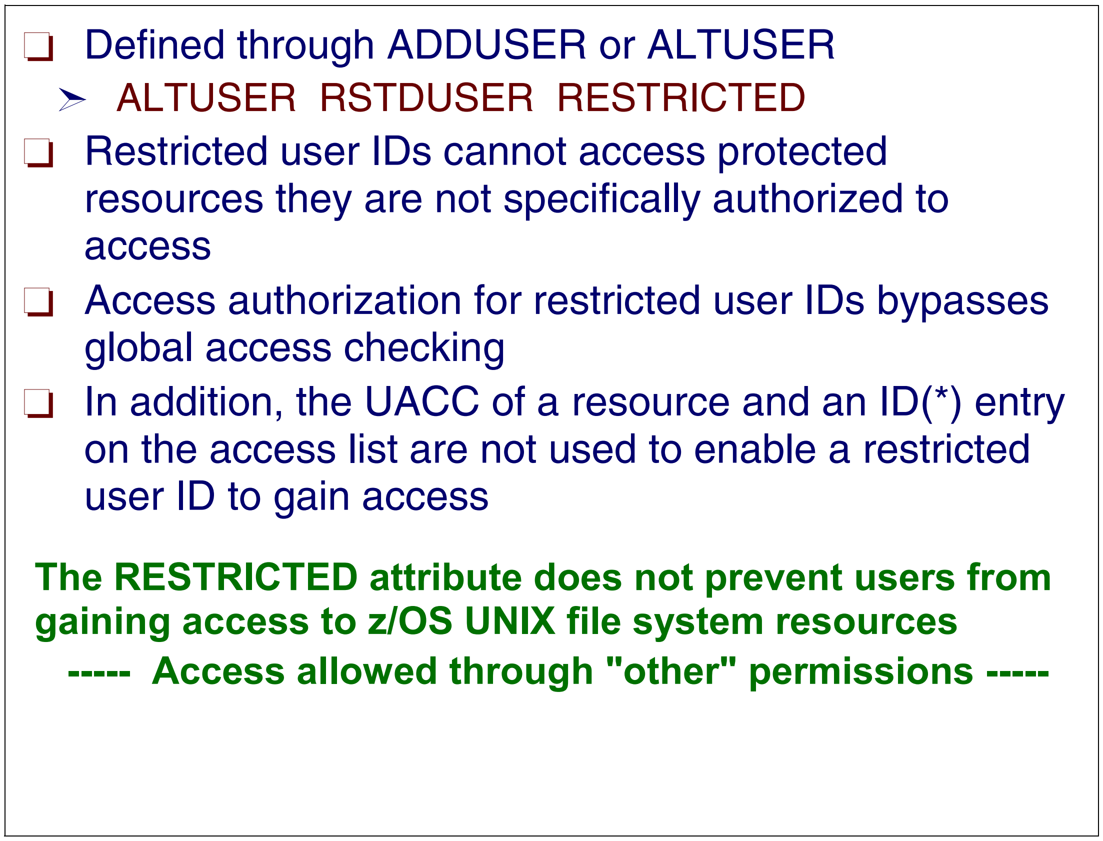
SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class if the caller's user ID is not either:

- UID(0)
- The owner of the file
- BPX.SUPERUSER

If the user executing the **chmod** command has at least READ authority to the resource, the user is authorized to change the file mode in the same manner as a user having UID(0).

This profile 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.

6.33 RACF RESTRICTED attribute



- Defined through ADDUSER or ALTUSER
  - ALTUSER RSTDUSER RESTRICTED
- Restricted user IDs cannot access protected resources they are not specifically authorized to access
- Access authorization for restricted user IDs bypasses global access checking
- In addition, the UACC of a resource and an ID(*) entry on the access list are not used to enable a restricted user ID to gain access

The RESTRICTED attribute does not prevent users from gaining access to z/OS UNIX file system resources

----- Access allowed through "other" permissions -----

Figure 6-33 Assigning the RESTRICTED attribute with RACF

You can define a restricted user ID by assigning the RESTRICTED attribute through the **ADDUSER** or **ALTUSER** command, as follows:

```
ALTUSER RSTDUSER RESTRICTED
```

User IDs with the RESTRICTED attribute cannot access protected resources they are not specifically authorized to access. Access authorization for restricted user IDs bypasses global access checking. In addition, the UACC of a resource and an ID(*) entry on the access list are not used to enable a restricted user ID to gain access.

However, the RESTRICTED attribute has no effect when a user accesses a z/OS UNIX file system resource; the file's “other” permission bits can allow access to users who are not explicitly authorized. To ensure that restricted users do not gain access to z/OS UNIX file system resources through other bits, you must use the new UNIXPRIV profile, RESTRICTED FILESYS.ACCESS.

6.34 z/OS UNIX file access checking

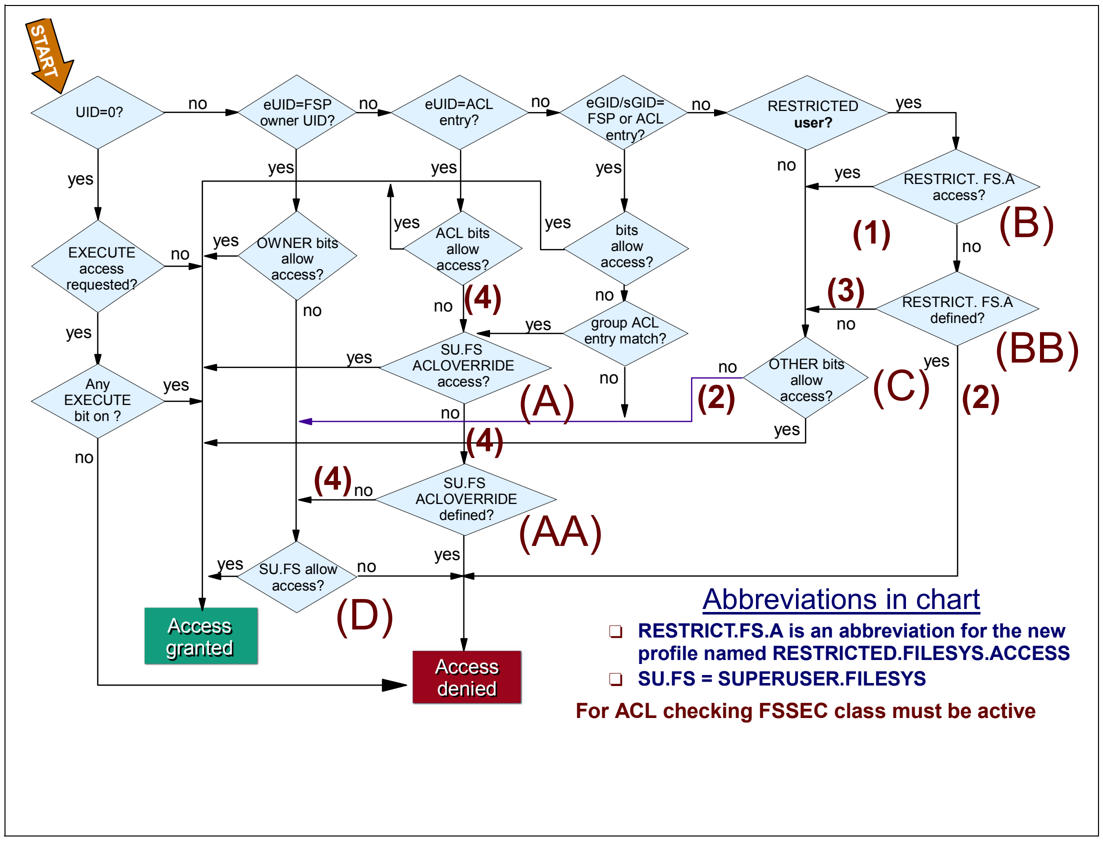


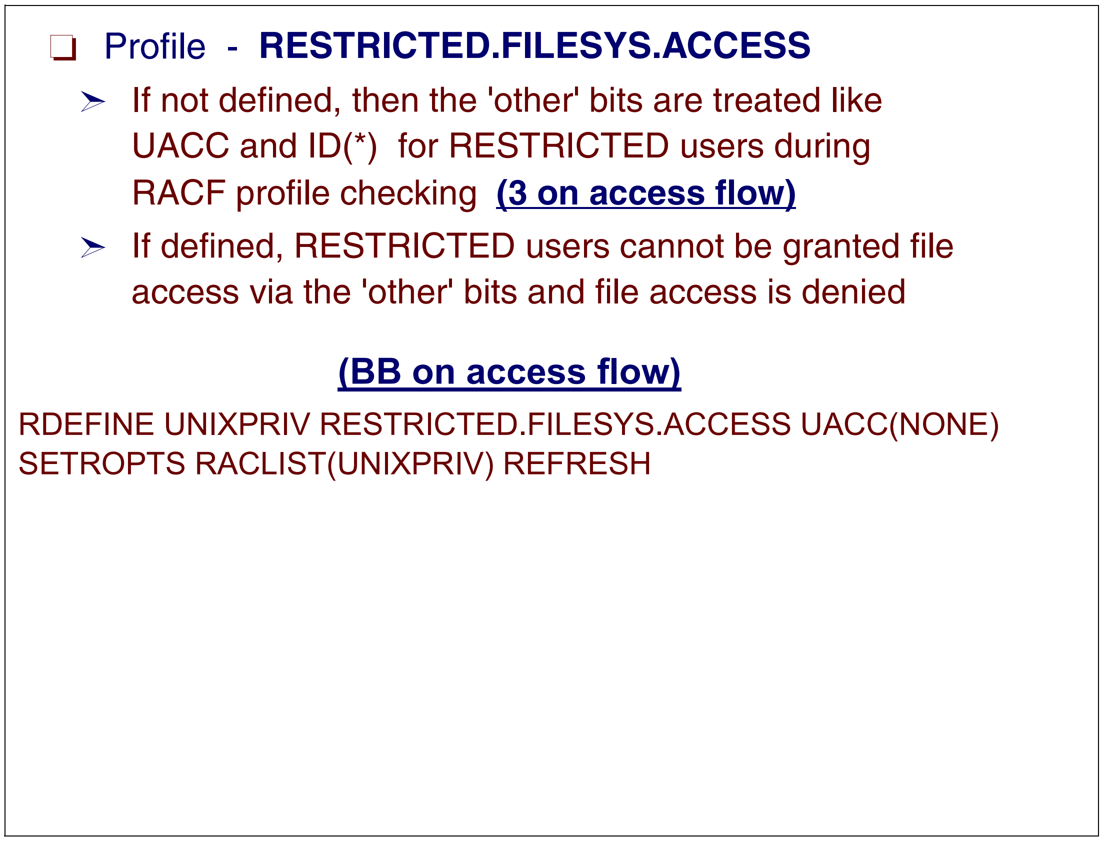
Figure 6-34 Authorization checking after introduction of UNIXPRIV profiles

The effective UID and effective GID of the process are used in determining access decisions. The only exception is that if file access is being tested, rather than requested, the real UID and GID are used instead of the effective UID and GID. The real and effective IDs are generally the same for a process, but if a set-uid or set-gid program is executed, they can be different.

- If the user is not system and is not a superuser, the permission bits and ACL (if one exists, and if the FSSEC class is active) for the file are checked to see if the access requested is allowed.
  - If the selected UID matches the owner UID of the file, the owner permission bits are checked.
  - If the UIDs don't match, the user ACL entries are checked. If the selected UID matches an ACL entry, the ACL entry bits are checked.
- If a matching ACL entry was not found for the user, the group bits and the group ACL entries are checked. The selected GID, and supplemental GIDs, are checked against the file owner GID and the group ACL entries, until a match is found which grants the requested access, or until all the GIDs have been checked.
  - If the GID matches the file owner GID, the file's "group" permission bits are checked. If the group bits allow the requested access, then access is granted.
  - If any of the user's supplemental GIDs match the file owner GID, the file's group permission bits are checked. If the group bits allow the requested access, then access is granted.

  - If no group bits access is allowed and the FSSEC class is active, and an ACL exists, and there is an ACL entry for any of the user's supplemental GIDs, then the permission bits of that ACL entry are checked. If at least one matching ACL entry was found for the GID, or any of the supplemental GIDs, then processing continues with the ACLOVERRIDE checking.
  - If no group ACL matches, then if the UNIXPRIV class is active, the SUPERUSER.FILESYS access is checked.
- SUPERUSER.FILESYS.ACLOVERRIDE is checked only when a user's access was denied by a matching ACL entry based on the user's UID or one of the user's GIDs. If the user's access was denied by the file's permission bits, SUPERUSER.FILESYS is checked.
- If no match was found, the other permission bits are checked, unless the user has the RESTRICTED attribute, the UNIXPRIV class is active, the resource named RESTRICTED.FILESYS.ACCESS is protected, and the user does not have at least READ access.

6.35 RESTRICTED user profile



- ❑ Profile - **RESTRICTED.FILESYS.ACCESS**
  - ➢ If not defined, then the 'other' bits are treated like UACC and ID(*) for RESTRICTED users during RACF profile checking **(3 on access flow)**
  - ➢ If defined, RESTRICTED users cannot be granted file access via the 'other' bits and file access is denied

(BB on access flow)

RDEFINE UNIXPRIV RESTRICTED.FILESYS.ACCESS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH

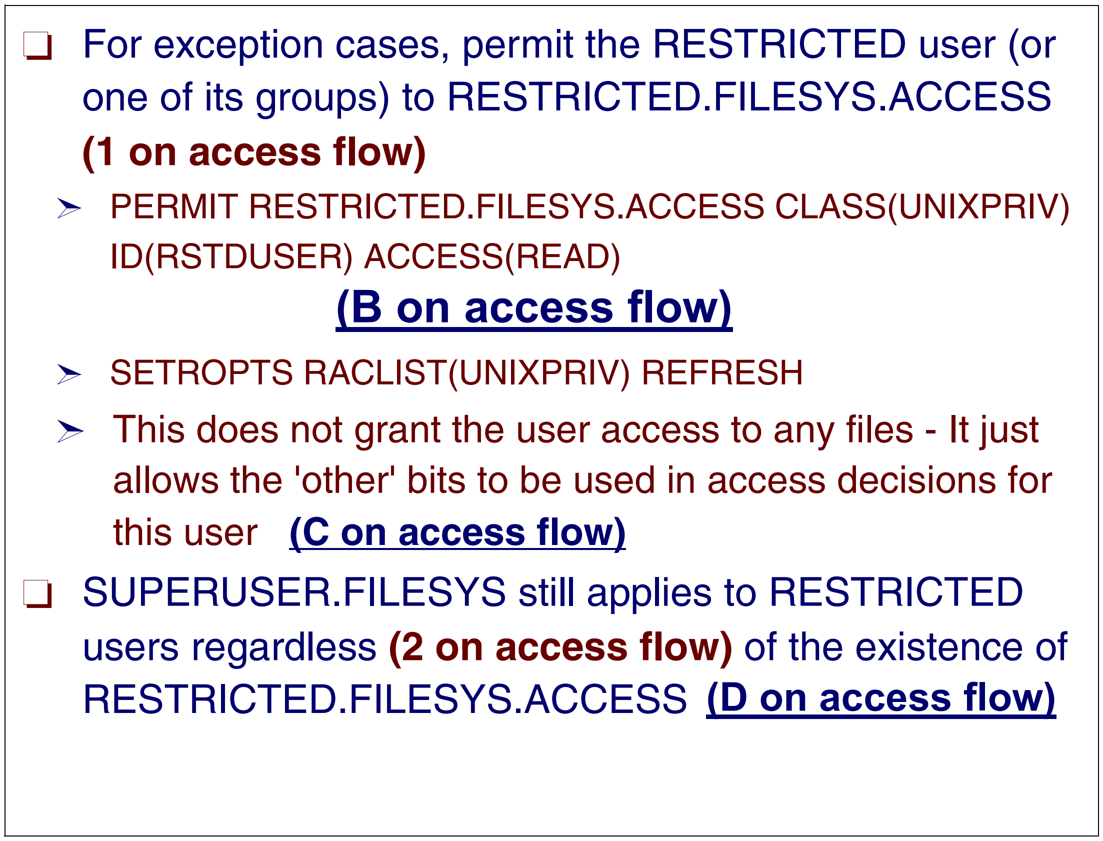
Figure 6-35 How the RESTRICTED user profile works

Users with the RESTRICTED attribute cannot access protected resources they are not specifically authorized to access. However, the RESTRICTED attribute has no effect when a user accesses a z/OS UNIX file system resource; therefore, if you do not define the RESTRICTED.FILESYS.ACCESS profile, then the file's “other” permission bits can allow access to users who are not explicitly authorized.

To ensure that restricted users do not gain access to z/OS UNIX file system resources through other bits, you must define the RACF profiles as follows:

```
RDEFINE UNIXPRIV RESTRICTED.FILESYS.ACCESS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

6.36 Restricted user access checking



- For exception cases, permit the RESTRICTED user (or one of its groups) to RESTRICTED.FILESYS.ACCESS **(1 on access flow)**
  - PERMIT RESTRICTED.FILESYS.ACCESS CLASS(UNIXPRIV) ID(RSTDUSER) ACCESS(READ)

    (B on access flow)
  - SETROPTS RACLIST(UNIXPRIV) REFRESH
  - This does not grant the user access to any files - It just allows the 'other' bits to be used in access decisions for this user **(C on access flow)**
- SUPERUSER.FILESYS still applies to RESTRICTED users regardless **(2 on access flow)** of the existence of RESTRICTED.FILESYS.ACCESS **(D on access flow)**

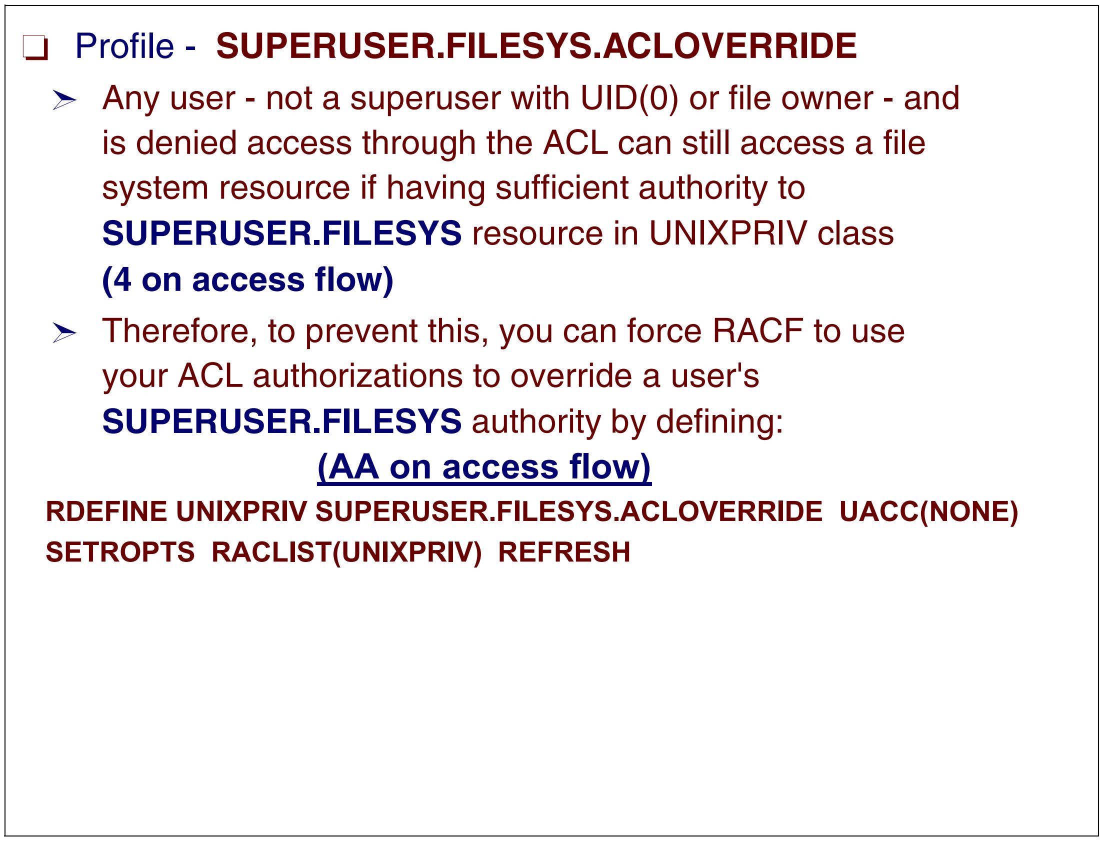
Figure 6-36 How the RESTRICTED user profile is used on the authority checking

If needed, grant exceptions to certain restricted users to allow them to gain access based on the file's other bits. Add those users, or one of their groups, to the access list with READ authority.

This does not grant the user access to any files. It just allows the other bits to be used in access decisions for this user.

Note: SUPERUSER.FILESYS still applies to RESTRICTED users regardless of the existence of the RESTRICTED.FILESYS.ACCESS profile.

6.37 Access checking with ACLs (1)



❑ Profile - **SUPERUSER.FILESYS.ACLOVERRIDE**

➢ Any user - not a superuser with UID(0) or file owner - and is denied access through the ACL can still access a file system resource if having sufficient authority to **SUPERUSER.FILESYS** resource in UNIXPRIV class **(4 on access flow)**

➢ Therefore, to prevent this, you can force RACF to use your ACL authorizations to override a user's **SUPERUSER.FILESYS** authority by defining: **(AA on access flow)**

RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH

Figure 6-37 Access checking with the ACLOVERRIDE profile

SUPERUSER.FILESYS.ACLOVERRIDE profile

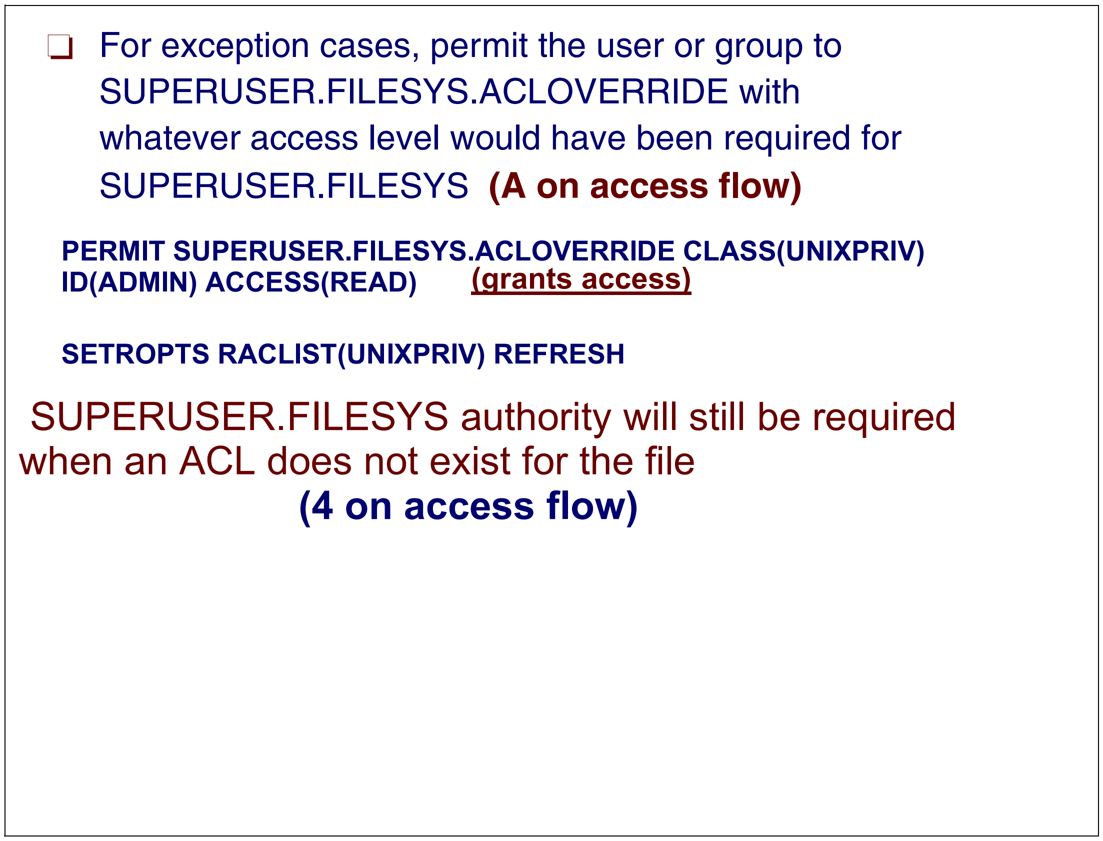
Any user who is not a superuser with UID(0) or the file owner, and who is denied access through the ACL, can still access a file system resource if the user has sufficient authority to the SUPERUSER.FILESYS resource in the UNIXPRIV class. Therefore, to prevent this, you can force RACF to use your ACL authorizations to override a user's SUPERUSER.FILESYS authority by defining the following profiles:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) RACLIST
```

"File authorization check summary" on page 174 shows the algorithm used by RACF in order to do authorization checking that now includes checking for profiles in the UNIXPRIV class for SUPER.FILESYS.ACLOVERRIDE, as shown at (AA).

Note: This describes the relationship between the existing SUPERUSER.FILESYS profile and the new SUPERUSER.FILESYS.ACLOVERRIDE profile. Either profile could get checked for a file; it depends upon the presence of an ACL for the file, and the contents of the ACL for granting access.

6.38 Access checking with ACLs (2)



- For exception cases, permit the user or group to SUPERUSER.FILESYS.ACLOVERRIDE with whatever access level would have been required for SUPERUSER.FILESYS **(A on access flow)**

PERMIT SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(ADMIN) ACCESS(READ) **(grants access)**

SETROPTS RACLIST(UNIXPRIV) REFRESH

SUPERUSER.FILESYS authority will still be required when an ACL does not exist for the file

(4 on access flow)

Figure 6-38 Access authority checking with the ACLOVERRIDE profile

For exception cases, permit the user or group to SUPERUSER.FILESYS.ACLOVERRIDE with whatever access level would have been required for SUPERUSER.FILESYS as follows:

```
PERMIT SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(ADMIN) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

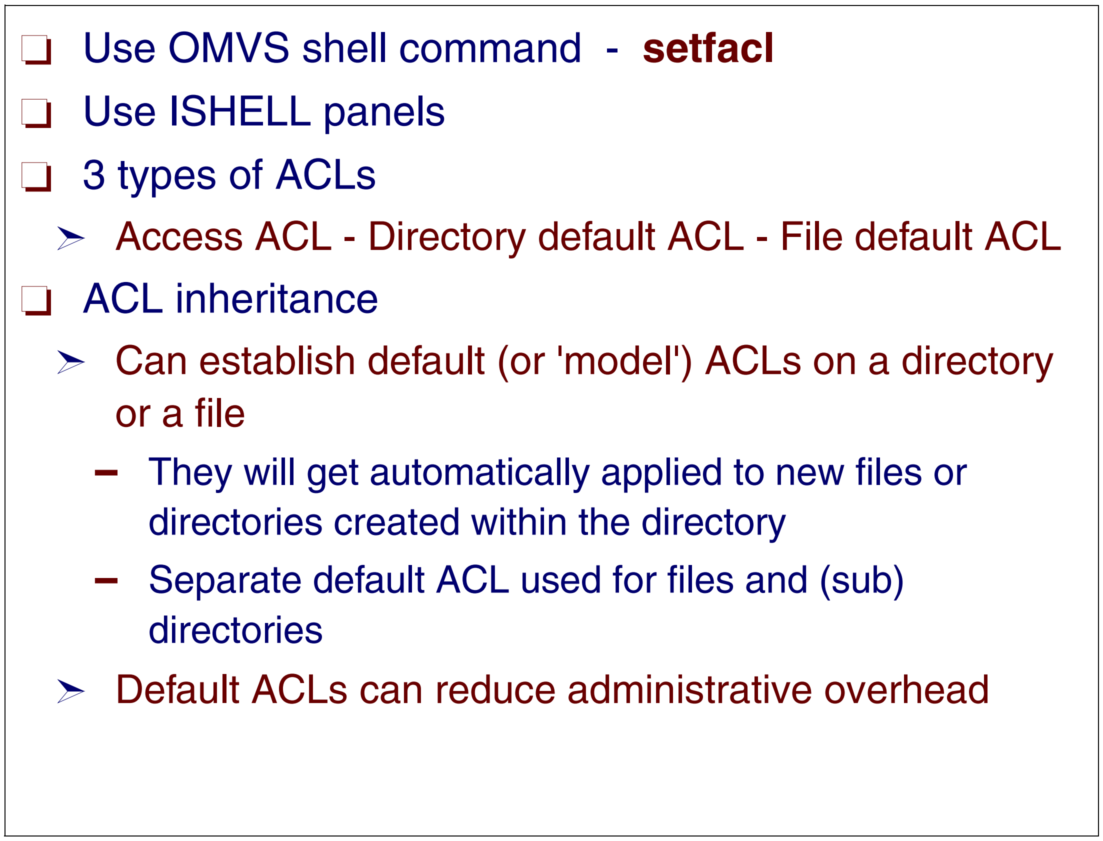
SUPERUSER.FILESYS authority is still checked when an ACL does not exist for the file. This should be done for administrators for whom you want total file access authority. That is, you do not want anyone to deny them access to a given file or directory by defining an ACL entry for them with limited, or no permission bit access.

This check is made at check (A) in the flow shown in the “File authorization check summary” on page 174.

SUPERUSER.FILESYS.ACLOVERRIDE is checked only when a user's access was denied by a matching ACL entry based on the user's UID or one of the user's GIDs. If the user's access was denied by the file's permission bits, SUPERUSER.FILESYS is checked.

Important: The intent of these new profiles is to allow ACLs to behave as much as possible like RACF profile access lists. The new profiles are provided to avoid changing the default behavior, as that could introduce compatibility issues with previous releases.

6.39 Create ACLs



- Use OMVS shell command - **setfacl**
- Use ISHELL panels
- 3 types of ACLs
  - Access ACL - Directory default ACL - File default ACL
- ACL inheritance
  - Can establish default (or 'model') ACLs on a directory or a file
    - They will get automatically applied to new files or directories created within the directory
    - Separate default ACL used for files and (sub) directories
  - Default ACLs can reduce administrative overhead

Figure 6-39 Types of ACLs that can be created

ACLs have existed on various UNIX platforms for many years, but with variations in the interfaces. ACL support in z/OSV1R3 is based on a POSIX standard (that was never approved) and other UNIX implementations. In the POSIX standard, two different ACLs are referenced as follows:

- Base ACL entries are permission bits (owner, group, other). It refers to the FSP.
- Extended ACL entries are ACL entries for individual users or groups, such as the permission bits that are stored with the file, not in RACF profiles.

Access control lists are introduced in z/OS V1R3 as a way to provide a greater granularity for access to z/OS UNIX files and directories. z/OS V1R3 provides support for ACLs to control access to files and directories by individual user (UID) and group (GID). ACLs are now created and checked by RACF. ACLs are created, modified, and deleted by using either of the following:

- `setfacl` shell command
- ISHELL interface

With the introduction of ACL support, when new files and directories are created in a file system, ACL inheritance is the process of automatically associating an ACL with a newly created object without requiring administrative action. ACL inheritance associates an ACL with the newly created file, myfile, without requiring administrative action. However, it is not always (and in fact, may be seldom) necessary to apply ACLs on every file or directory within a subtree. If you have a requirement to grant access to an entire subtree (for example, a subtree specific to a given application), then access can be established at the top directory. If a given user or group does not have search access to the top directory, then no files within the subtree will be accessible, regardless of the permission bit settings or ACL contents associated with these files. The user or group will still need permission to the files within the directory subtree where appropriate. If this is already granted by the “group” or “other” bits, then no ACLs are necessary below the top directory.

Note: When defining ACLs, it is recommended to place ACLs on directories, rather than on each file in a directory.

The phrases “default ACL” and “model ACL” are used interchangeably throughout z/OS UNIX documentation. Other systems that support ACLs have default ACLs that are essentially the same as the directory default ACLs in z/OS UNIX. According to the X/Open UNIX 95 specification, additional access control mechanisms may 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.

To create an ACL for a file, you must have one of the following security access controls:

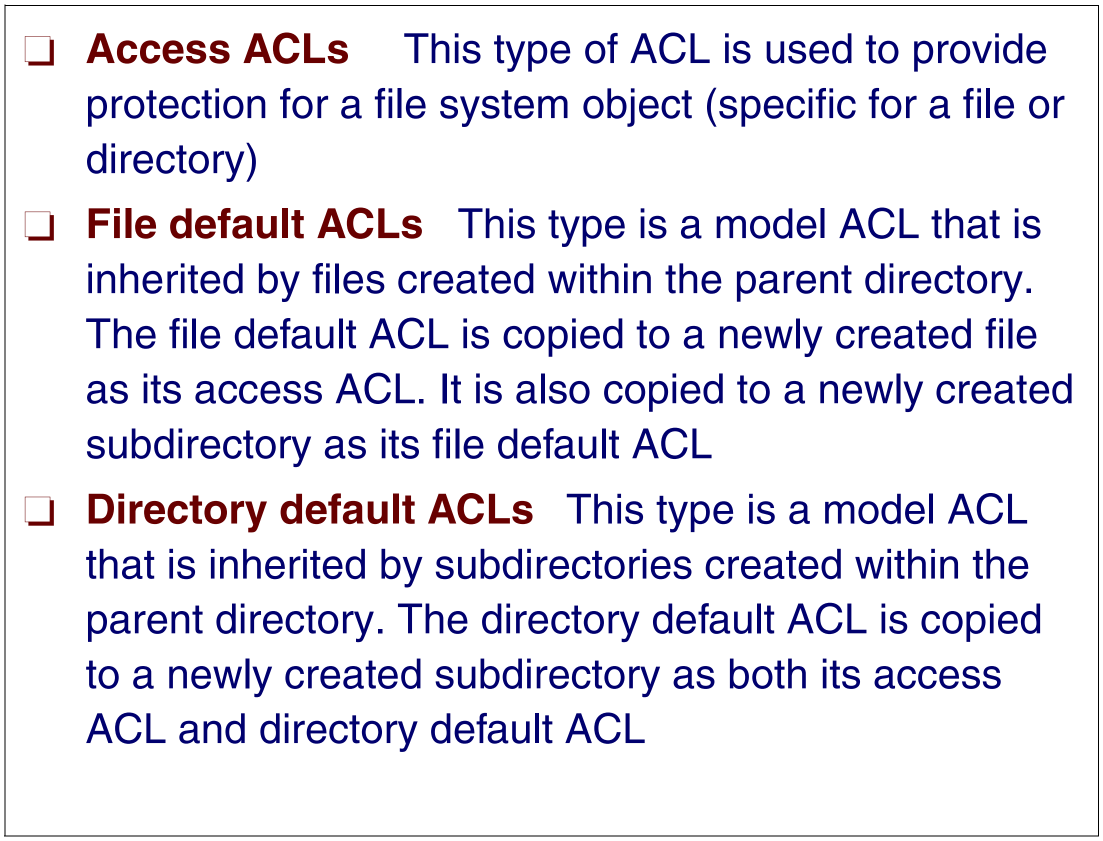
- Be the file owner
- BPX.SUPERUSER
- Have superuser authority (UID=0)
- Have READ access to the SUPERUSER.FILESYS.CHANGEPERMS profile in the UNIXPRIV class

Note: The RACF UNIXPRIV class was introduced in OS/390 V2R8. It allows you to 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 grant specific superuser privileges to users who do not have superuser authority (UID=0). This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

To activate the use of ACLs in z/OS UNIX file authority checks, the following RACF command needs to be run to activate the new RACF class FSSEC:

```
SETROPTS CLASSACT(FSSEC)
```

6.40 ACL types



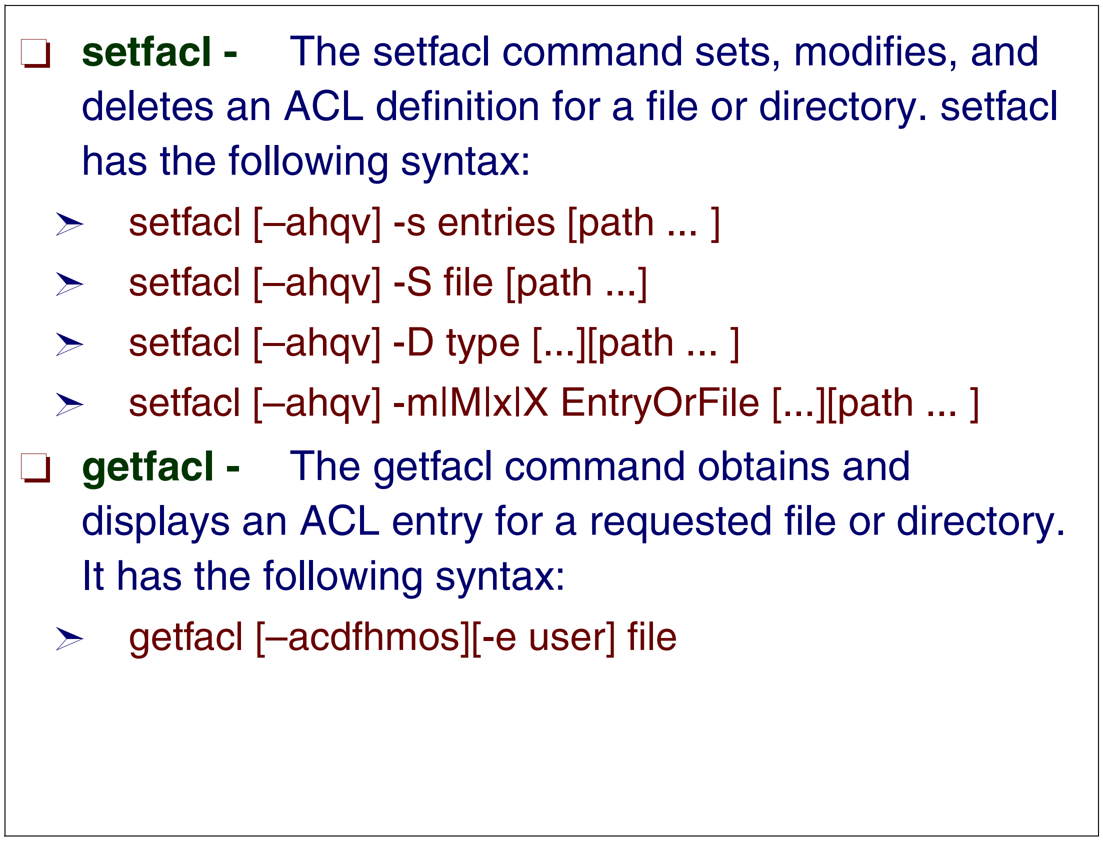
- **Access ACLs** This type of ACL is used to provide protection for a file system object (specific for a file or directory)
- **File default ACLs** This type is a model ACL that is inherited by files created within the parent directory. The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL
- **Directory default ACLs** This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL

Figure 6-40 Types of ACLs that can be created

To reduce administrative overhead, three types of ACLs (extended ACLs) are defined in order to have the capability to inherit ACLs to newly created files and directories, as follows:

Access ACLs	This type of ACL is used to provide protection for a file system object (specific for a file or directory).
File default ACLs	This type is a model ACL that is inherited by files created within the parent directory. 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.
Directory default ACLs	This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL.

6.41 OMVS shell commands for ACLs



- **setfacl -** The setfacl command sets, modifies, and deletes an ACL definition for a file or directory. setfacl has the following syntax:
  - setfacl [–ahqv] -s entries [path ...]
  - setfacl [–ahqv] -S file [path ...]
  - setfacl [–ahqv] -D type [...][path ...]
  - setfacl [–ahqv] -m|M|x|X EntryOrFile [...][path ...]
- **getfacl -** The getfacl command obtains and displays an ACL entry for a requested file or directory. It has the following syntax:
  - getfacl [–acdfhmos][-e user] file

Figure 6-41 OMVS shell commands to create and display ACLs

The new shell commands, **`setfacl`** and **`getfacl,`** have been introduced in order to create, modify, and display ACL entries specified by the path.

Use ACLs to control access to files and directories by individual user and group. 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.

6.42 Create ACLs for a specific directory

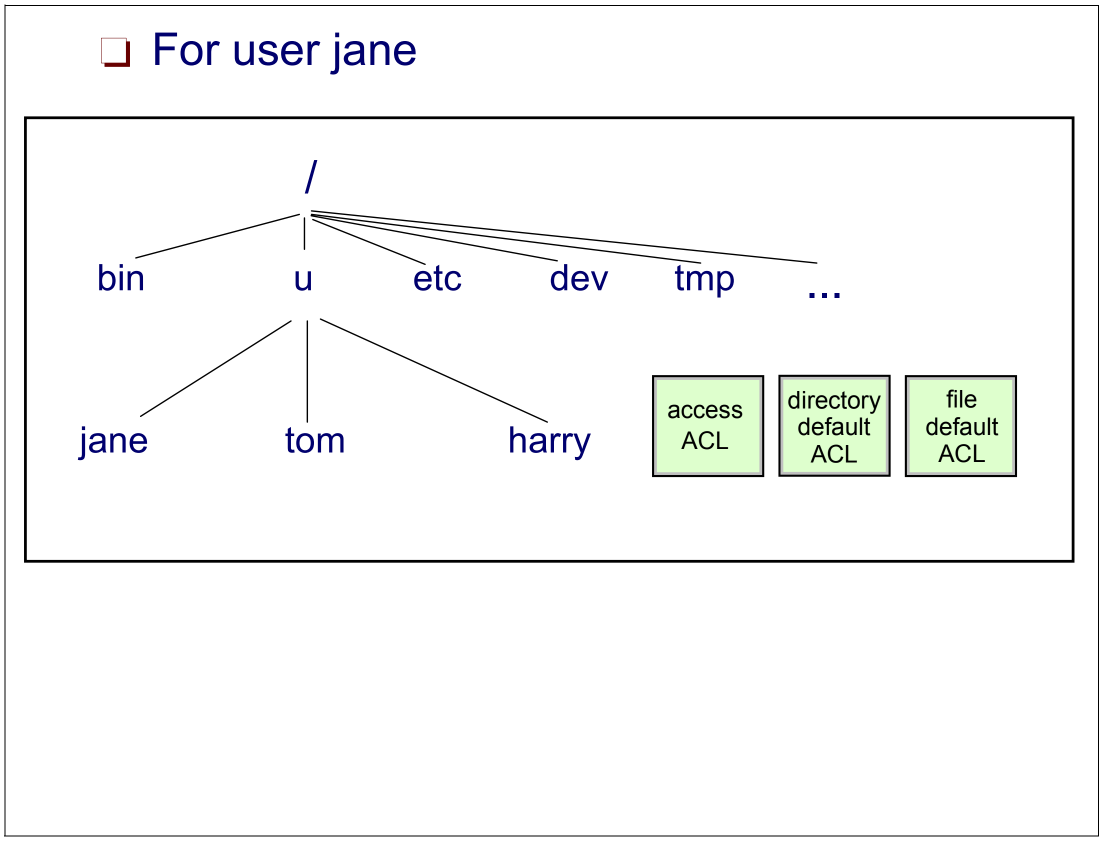
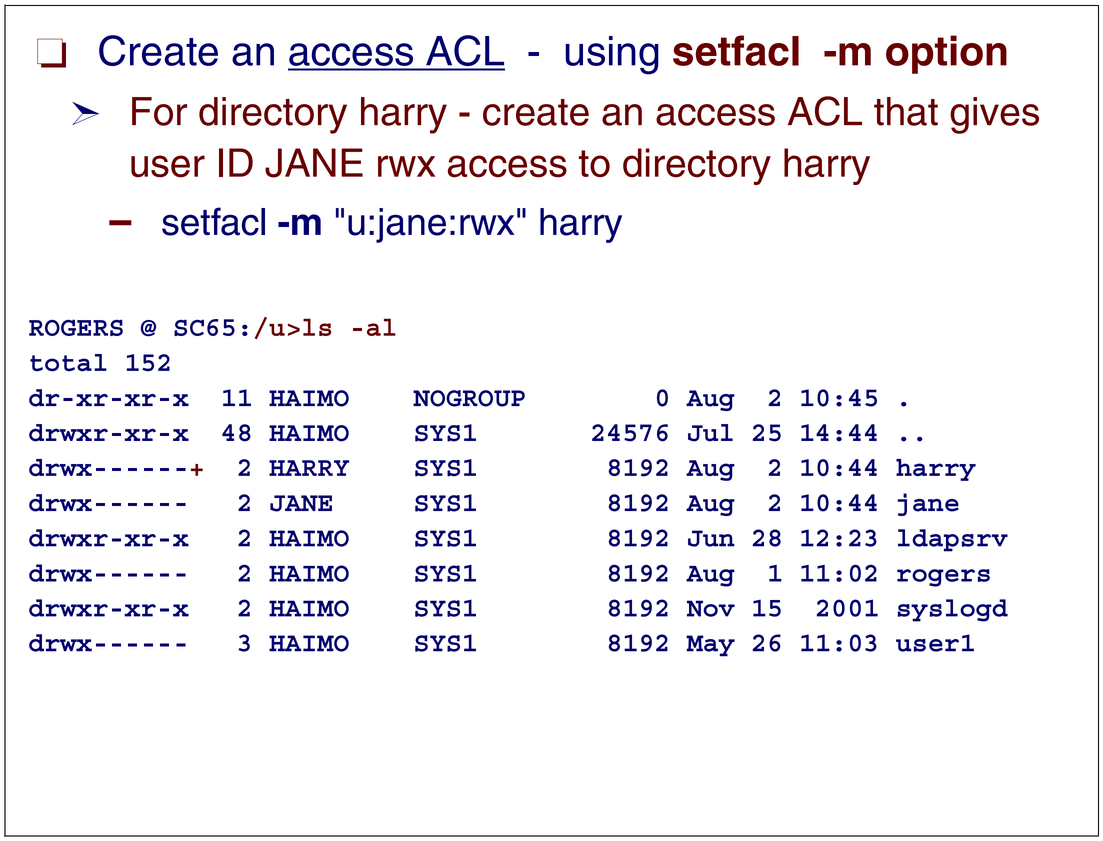


Figure 6-42 Creating ACLs for a specific directory

The next few figures show how to create the three ACLs shown in Figure 6-42 that give *user jane* access to *directory harry* and any directories and files defined under directory harry.

6.43 Create an access ACL



- Create an access ACL - using **setfacl -m option**
  - For directory harry - create an access ACL that gives user ID JANE rwx access to directory harry
    - setfacl **-m** "u:jane:rwx" harry

```
ROGERS @ SC65:/u>ls -al
total 152
dr-xr-xr-x  11 HAIMO    NOGROUP        0 Aug  2 10:45 .
drwxr-xr-x  48 HAIMO    SYS1       24576 Jul 25 14:44 ..
drwx------+  2 HARRY    SYS1        8192 Aug  2 10:44 harry
drwx------   2 JANE     SYS1        8192 Aug  2 10:44 jane
drwxr-xr-x   2 HAIMO    SYS1        8192 Jun 28 12:23 ldapsrv
drwx------   2 HAIMO    SYS1        8192 Aug  1 11:02 rogers
drwxr-xr-x   2 HAIMO    SYS1        8192 Nov 15  2001 syslogd
drwx------   3 HAIMO    SYS1        8192 May 26 11:03 user1
```

Figure 6-43 Create an access ACL for directory HARRY

These ACLs are used to provide protection for a file system directory or file. When you are setting the access ACL, the ACL entries must consist of three required base ACL entries that correspond to the file permission bits. The ACL entries must also consist of zero or more extended ACL entries, which will allow a greater level of granularity when controlling access. The permissions for base entries must be in absolute form.

With the `setfacl` command, you use the **-s** option to create an ACL. You must create the entire ACL, which includes the base ACL and extended ACL. The base ACL (permission bits) are indicated by omitting user or group qualifiers.

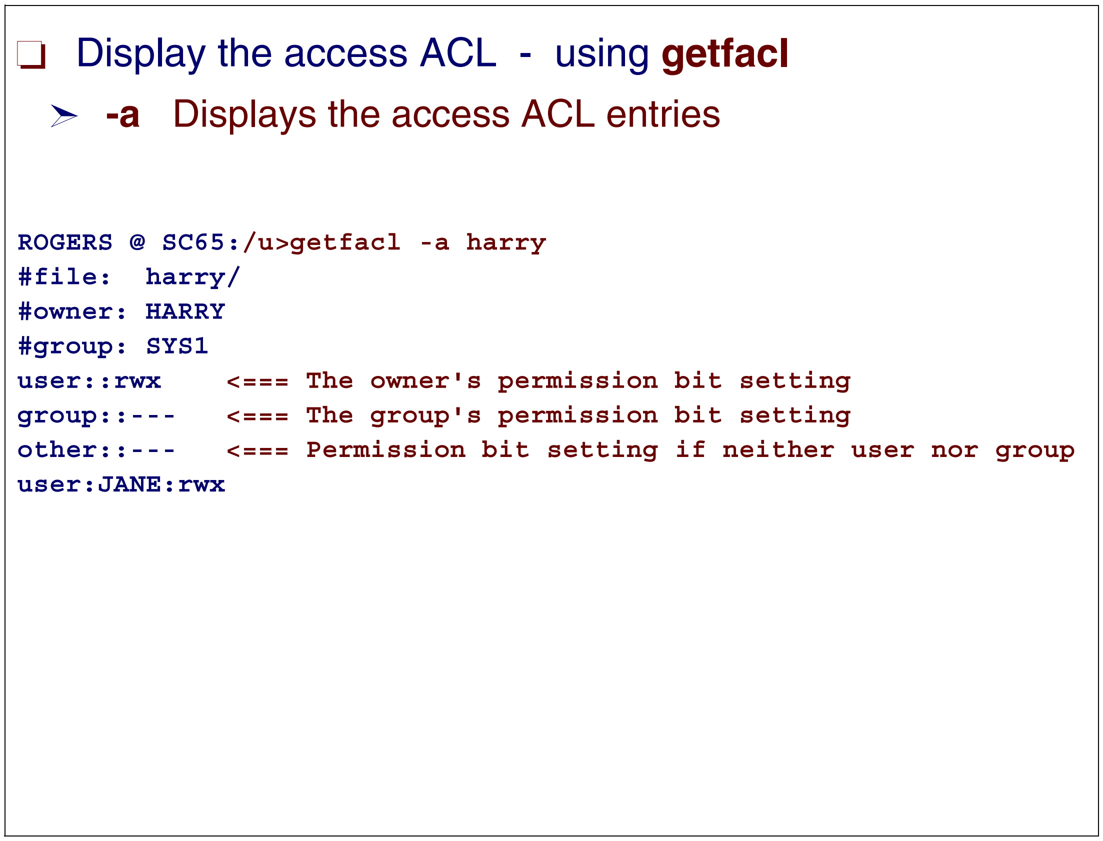
The following command creates an access ACL that gives user ID JANE rwx access to directory harry:

```
ROGERS @ SC65:/u>setfacl -m “u:jane:rwx” harry
```

Note: When you are *setting* the access ACL, the ACL entries must consist of the three required base ACL entries that correspond to the file permission bits (`u::rwx`). The ACL entries must also consist of zero or more extended ACL entries (`g::---,o::---,u:jane:rwx`), which will allow a greater level of granularity when controlling access. The permissions for base entries must be in absolute form.

As shown in Figure 6-43, issuing the `ls -al` command, the **+** sign following the permission bits for directory harry indicates that an ACL exists for that directory.

6.44 Display the access ACL



❑ Display the access ACL - using **getfacl**

➢ **-a** Displays the access ACL entries

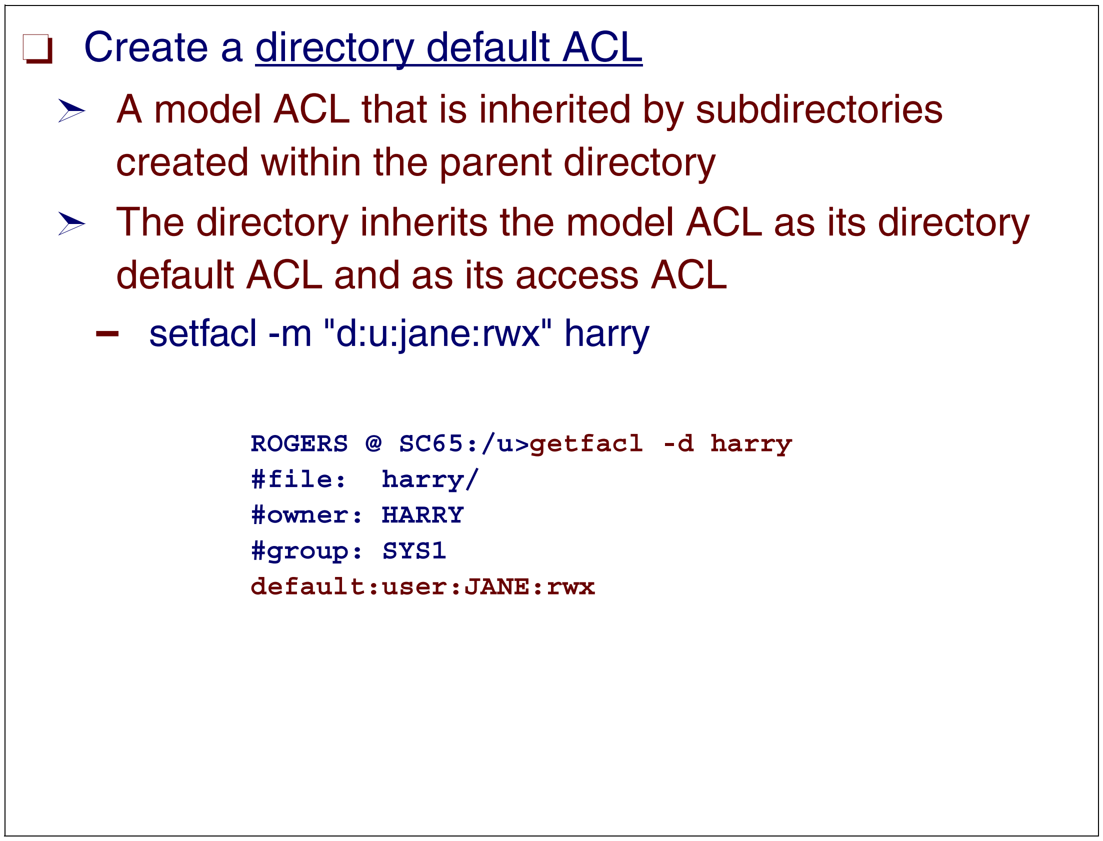
```
ROGERS @ SC65:/u>getfacl -a harry
#file:  harry/
#owner: HARRY
#group: SYS1
user::rwx    <=== The owner's permission bit setting
group::---   <=== The group's permission bit setting
other::---   <=== Permission bit setting if neither user nor group
user:JANE:rwx
```

Figure 6-44 Displaying the created ACL using the **`getfacl`** *command*

The **`getfacl`** command displays the comment header, base ACL entries, and extended ACL entries, if there are any, for each file that is specified. It also resolves symbolic links. You can specify whether to display access, file default, or directory default. You can also change the default display format. The output can be used as input to **`setfacl`**.

-a Displays the access ACL entries. This is the default if `-a`, `-d`, or `-f` is not specified. Figure 6-44 displays the ACL entry just created for directory harry.

6.45 Create a directory default ACL



- Create a directory default ACL
  - A model ACL that is inherited by subdirectories created within the parent directory
  - The directory inherits the model ACL as its directory default ACL and as its access ACL
    - setfacl -m "d:u:jane:rwx" harry

```
ROGERS @ SC65:/u>getfacl -d harry
#file:  harry/
#owner: HARRY
#group: SYS1
default:user:JANE:rwx
```

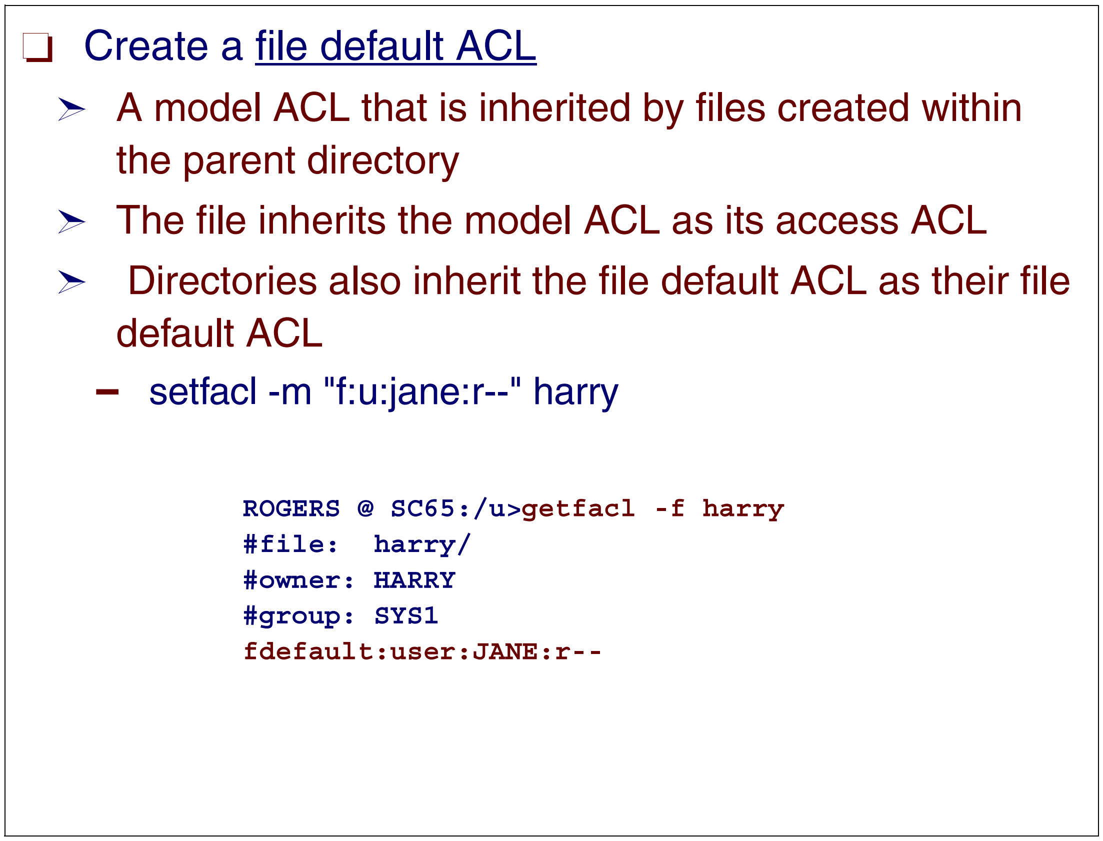
Figure 6-45 Creating a directory default ACL for directory HARRY

Directory default ACLs are model ACLs that are inherited by subdirectories created within the parent directory. The directory inherits the model ACL as its directory default ACL and as its access ACL when a new directory is created under directory harry.

For directory harry, the following command creates a directory default ACL that gives user jane rwx access in a directory default ACL for directory harry:

```
ROGERS @ SC65:/u>setfacl -m "d:u:jane:rwx" harry
```

6.46 Create a file default ACL



- Create a file default ACL
  - A model ACL that is inherited by files created within the parent directory
  - The file inherits the model ACL as its access ACL
  - Directories also inherit the file default ACL as their file default ACL
    - setfacl -m "f:u:jane:r--" harry

```
ROGERS @ SC65:/u>getfacl -f harry
#file:  harry/
#owner: HARRY
#group: SYS1
fdefault:user:JANE:r--
```

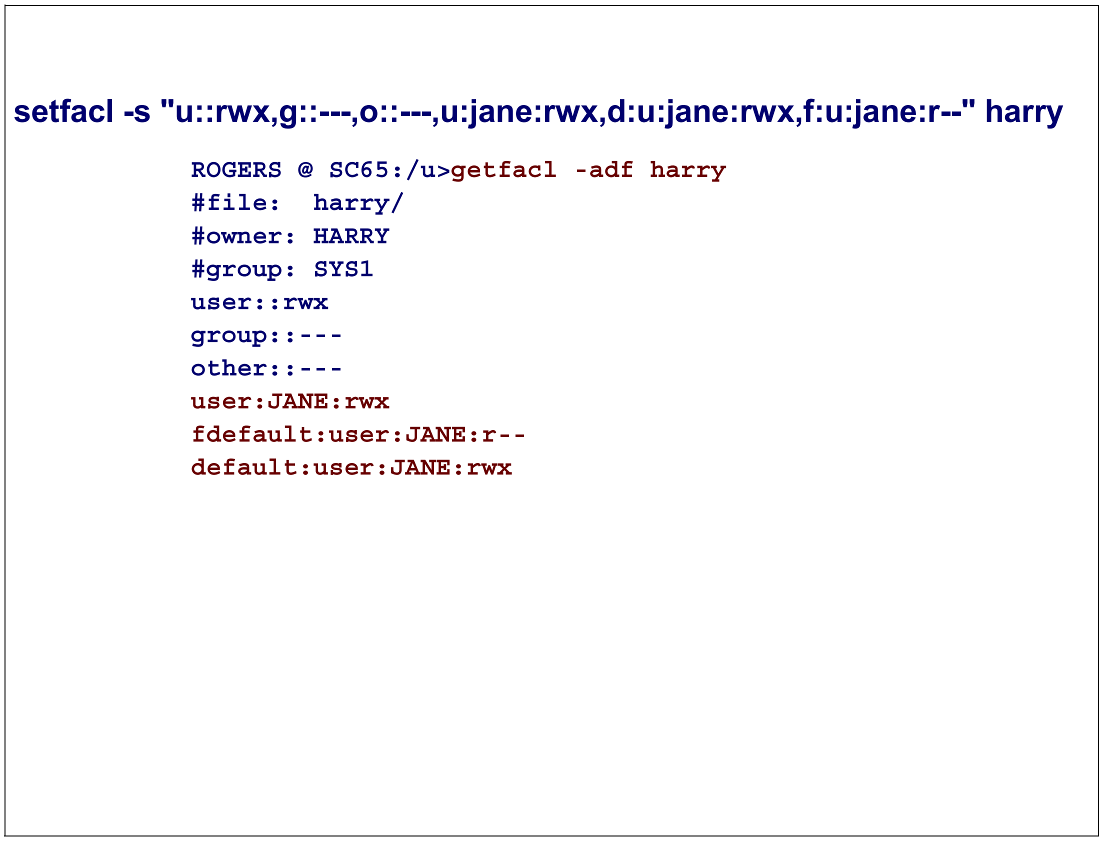
Figure 6-46 Create a file default ACL for directory HARRY

File default ACLs are model ACLs that are inherited by files created within the parent directory. The file inherits the model ACL as its access ACL. Directories also inherit the file default ACL as their file default ACL.

For directory harry, the following command creates a file default ACL that gives user jane r-- access in a file default ACL for directory harry:

```
ROGERS @ SC65:/u>setfacl -m "f:u:jane:r--" harry
```

6.47 Creating all ACL types



setfacl -s "u::rwx,g::---,o::---,u:jane:rwx,d:u:jane:rwx,f:u:jane:r--" harry

```
ROGERS @ SC65:/u>getfacl -adf harry
#file:  harry/
#owner: HARRY
#group: SYS1
user::rwx
group::---
other::---
user:JANE:rwx
fdefault:user:JANE:r--
default:user:JANE:rwx
```

Figure 6-47 Creating all three ACLs and specifying the permission settings

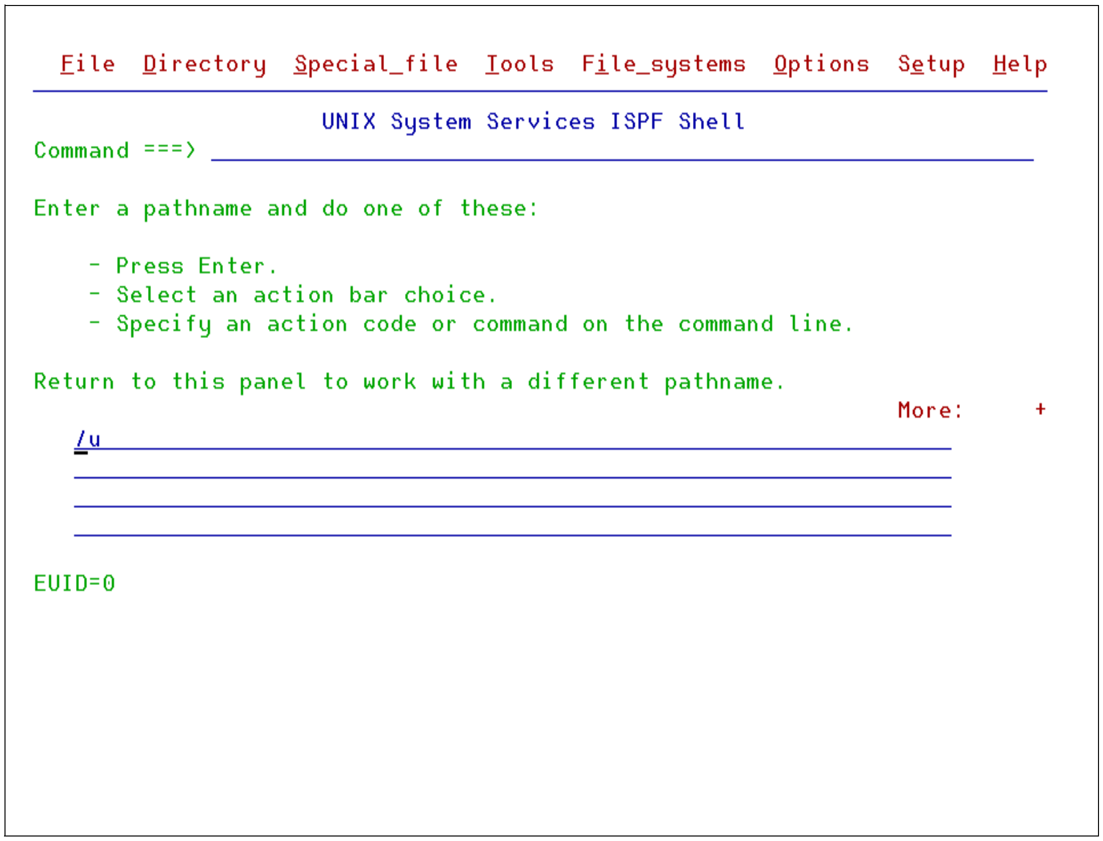
If you want to create all three ACLs for the directory, you can issue just one command for ACLs, and by using -s, you can specify the current permission bit settings as follows:

```
setfacl -s "u::rwx,g::---,o::---,u:jane:rwx,d:u:jane:rwx,f:u:jane:r--" harry
```

To display the ACLs just created by the **`setfacl`** command, issue the following command:

```
getfacl -adf harry
```

6.48 Using the ISHELL panel



```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 ─────────────────────────────────────────────────────────────────────────
                          UNIX System Services ISPF Shell
Command ===> ____________________________________________________________

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                             More:     +
   /u_________________________________________________________________
   ___________________________________________________________________
   ___________________________________________________________________
   ___________________________________________________________________

EUID=0
```

Figure 6-48 Using the ISHELL panel to create ACLs

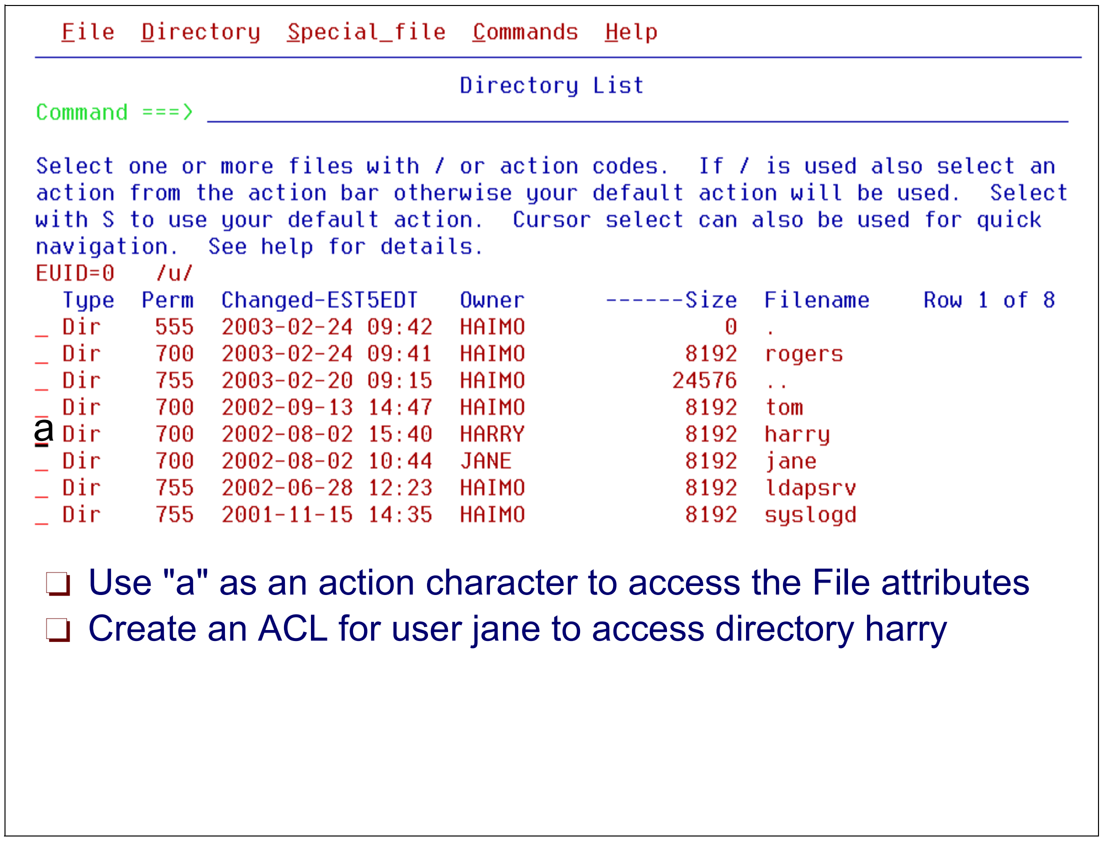
With the ISPF shell, a user or system programmer can use ISPF dialogs instead of shell commands to perform many tasks, especially those related to file systems and files. An ordinary user can use the ISPF shell to work with:

- Directories
- Regular files
- FIFO special files
- Symbolic links, including external links

You can also run shell commands, REXX programs, and C programs from the ISPF shell. The ISPF shell can direct stdout and stderr only to an HFS file, not to your terminal. If it has any contents, the file is displayed when the command or program completes.

Note: New features have been added to improve the ISHELL functionality on z/OS V1R3. The ISHELL is a user interface that allows users to work with menus rather than sometimes cryptic commands. In z/OS V1R3, many new features that have been requested over the years to improve ISHELL functionality and panel navigation are implemented. These new features are a significant upgrade to the ISHELL, and many of them are part of the Directory List option which lists files in a particular directory. Other enhancements include sorting and highlighting support, as well as easy ways to access information. The effective user ID(EUID) is displayed on the panel so you can know the authority you have at any particular moment. In the figure, EUID=0 indicates that the current UID that is accessing the ISHELL is a superuser.

6.49 Create an access ACL using ISHELL



```
  File  Directory  Special_file  Commands  Help

                              Directory List
Command ===> ____________________________________________________________________

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /u/
  Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 8
_ Dir    555  2003-02-24 09:42  HAIMO               0  .
_ Dir    700  2003-02-24 09:41  HAIMO            8192  rogers
_ Dir    755  2003-02-20 09:15  HAIMO           24576  ..
_ Dir    700  2002-09-13 14:47  HAIMO            8192  tom
a Dir    700  2002-08-02 15:40  HARRY            8192  harry
_ Dir    700  2002-08-02 10:44  JANE             8192  jane
_ Dir    755  2002-06-28 12:23  HAIMO            8192  ldapsrv
_ Dir    755  2001-11-15 14:35  HAIMO            8192  syslogd
```

- Use "a" as an action character to access the File attributes
- Create an ACL for user jane to access directory harry

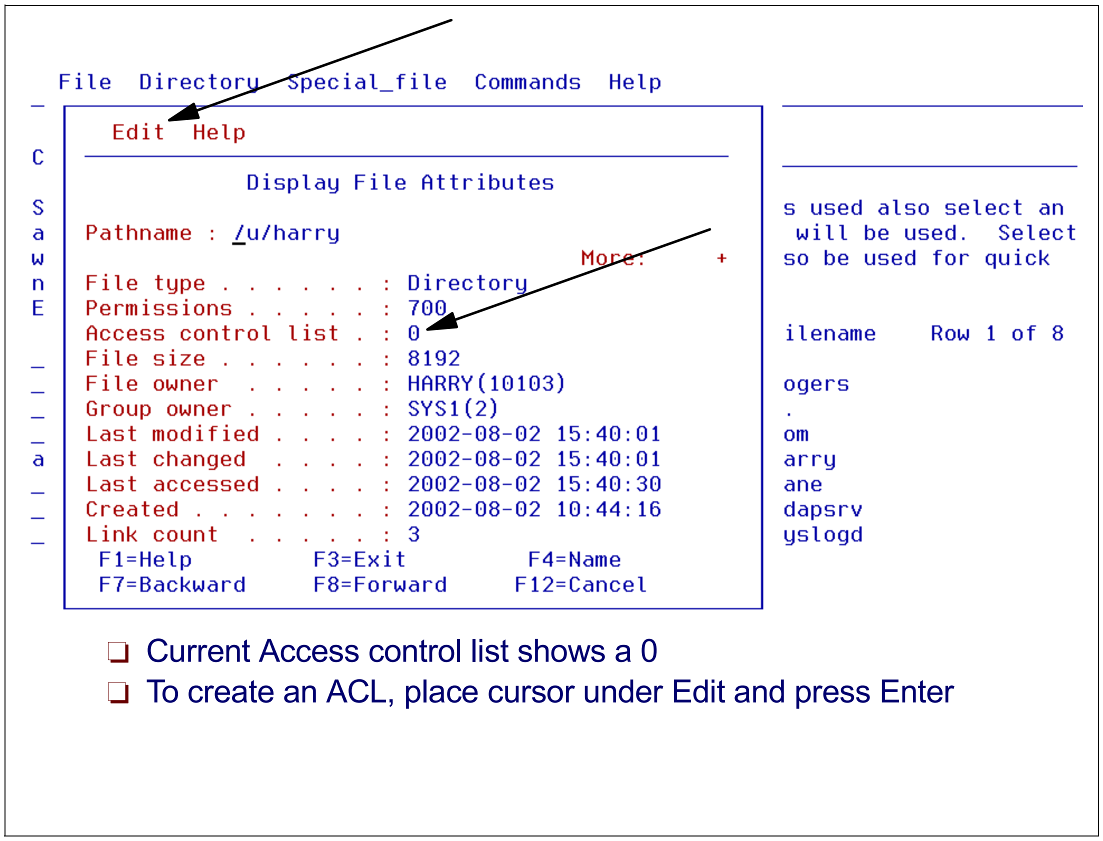
Figure 6-49 Creating an access ACL using the ISHELL

Figure 6-49 displays the Directory List once you have entered the ISHELL, typed `/u` on the command line, and pressed Enter.

By placing an "**a**" action code for directory harry, the **File Attribute** panel is displayed. This is the first step to create ACLs.

We are going to create an ACL that gives user JANE access to directory HARRY.

6.50 File attributes panel for /u/harry

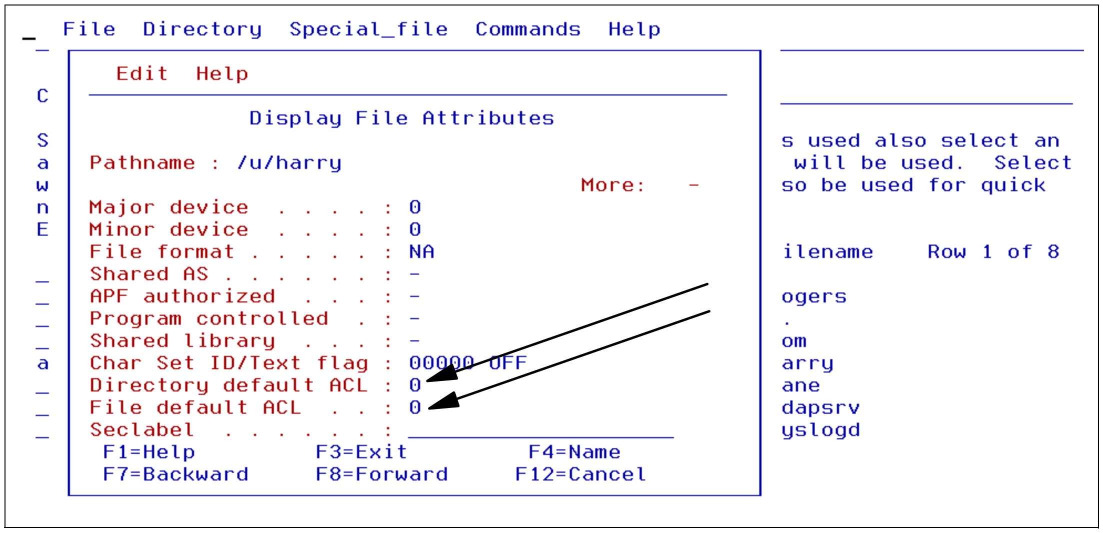


```
  File  Directory  Special_file  Commands  Help
_
     Edit  Help
C
               Display File Attributes
S                                                   s used also select an
a    Pathname : /u/harry                             will be used.  Select
w                                       More:     +  so be used for quick
n    File type . . . . . . : Directory
E    Permissions . . . . . : 700
     Access control list . : 0                       ilename    Row 1 of 8
_    File size . . . . . . : 8192
_    File owner  . . . . . : HARRY(10103)            ogers
_    Group owner . . . . . : SYS1(2)                 .
_    Last modified . . . . : 2002-08-02 15:40:01     om
a    Last changed  . . . . : 2002-08-02 15:40:01     arry
_    Last accessed . . . . : 2002-08-02 15:40:30     ane
_    Created . . . . . . . : 2002-08-02 10:44:16     dapsrv
_    Link count  . . . . . : 3                       yslogd
      F1=Help         F3=Exit         F4=Name
      F7=Backward     F8=Forward     F12=Cancel
```

- Current Access control list shows a 0
- To create an ACL, place cursor under Edit and press Enter

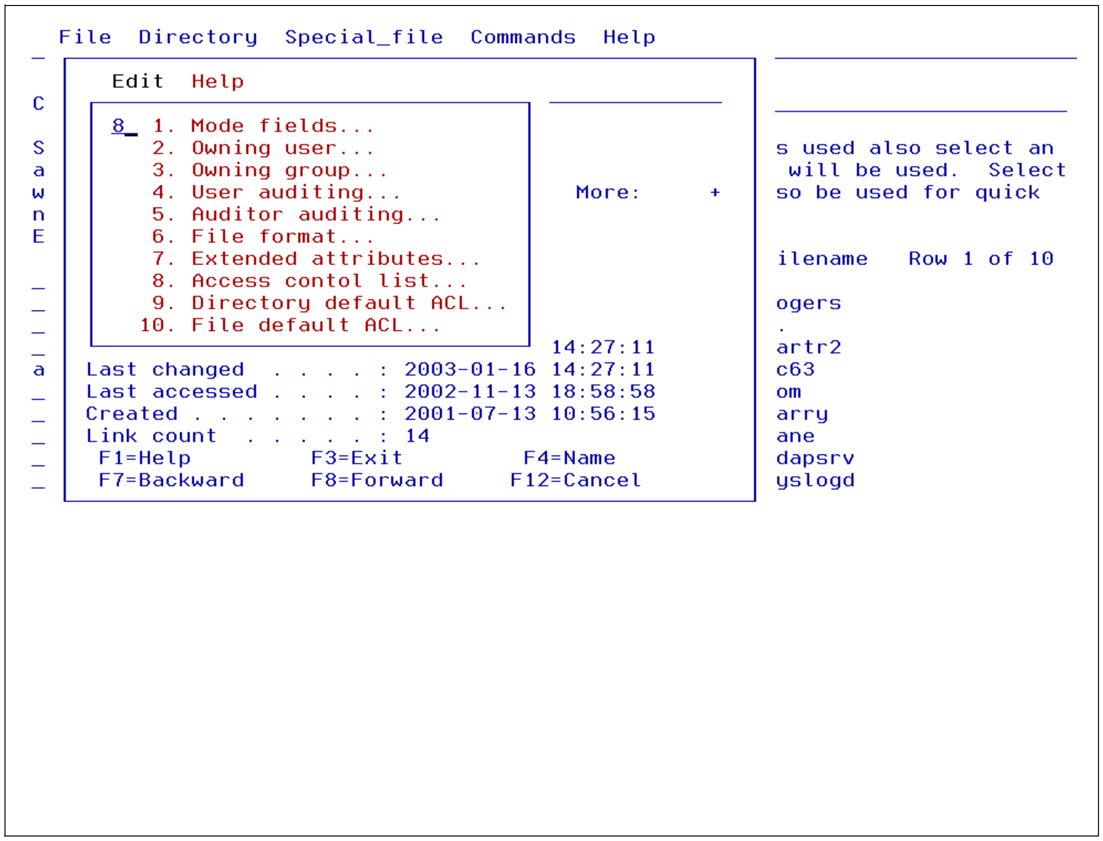
Figure 6-50 Display File Attribute panel showing no ACL exists

If you scroll forward twice, you can see if a Directory Default ACL or File Default ACL is defined, as shown in the following panel. These two fields (Directory Default and File Default ACL) only apply to directory files.



```
_  File  Directory  Special_file  Commands  Help
_
     Edit  Help
C
               Display File Attributes
S                                                   s used also select an
a    Pathname : /u/harry                             will be used.  Select
w                                       More:   -    so be used for quick
n    Major device  . . . . : 0
E    Minor device  . . . . : 0
     File format . . . . . : NA                      ilename    Row 1 of 8
_    Shared AS . . . . . . : -
_    APF authorized  . . . : -                       ogers
_    Program controlled  . : -                       .
_    Shared library  . . . : -                       om
a    Char Set ID/Text flag : 00000 OFF               arry
_    Directory default ACL : 0                       ane
_    File default ACL  . . : 0                       dapsrv
_    Seclabel  . . . . . . : ____________________    yslogd
      F1=Help         F3=Exit         F4=Name
      F7=Backward     F8=Forward     F12=Cancel
```

6.51 Select Option 8 to create an access ACL

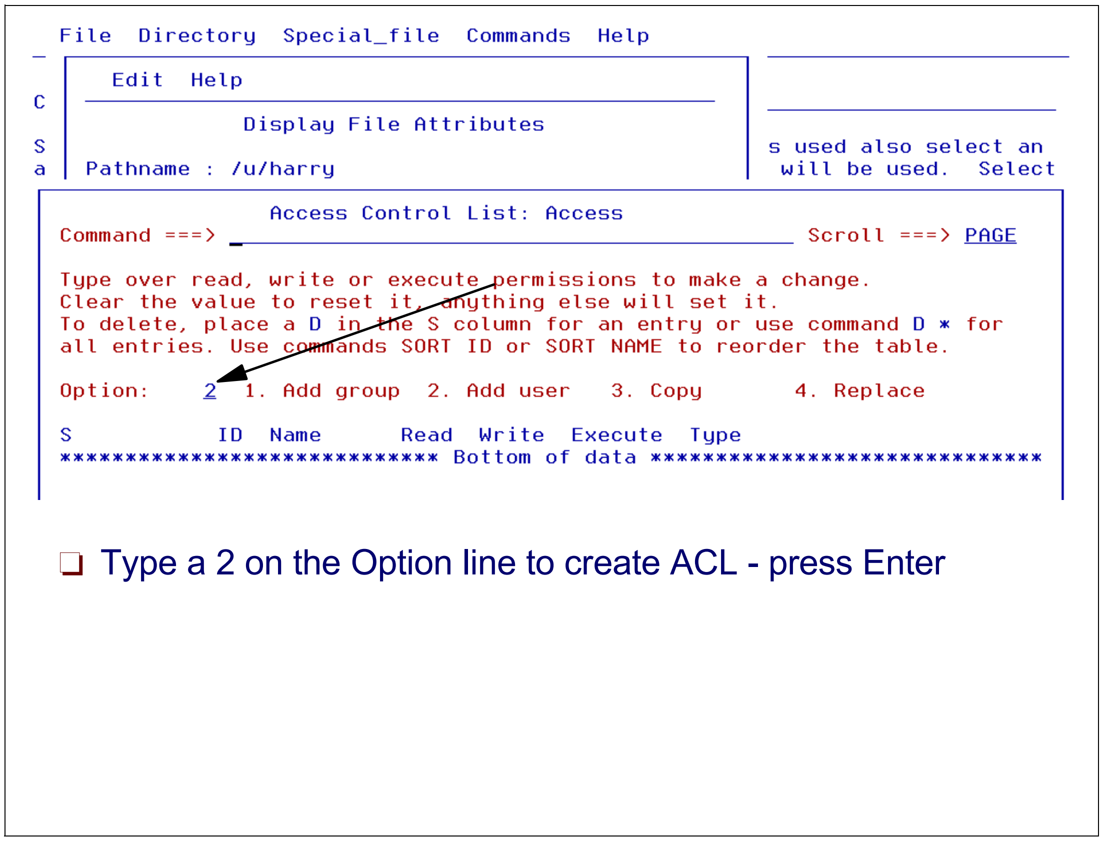


```
  File  Directory  Special_file  Commands  Help
_  ┌─────────────────────────────────────────────────┐  ______________________
   │  Edit  Help                                     │
C  │ ┌──────────────────────────────────┐ _______   │  ______________________
   │ │ 8_  1. Mode fields...            │           │
S  │ │     2. Owning user...            │           │  s used also select an
a  │ │     3. Owning group...           │           │   will be used.  Select
w  │ │     4. User auditing...          │ More:   + │  so be used for quick
n  │ │     5. Auditor auditing...       │           │
E  │ │     6. File format...            │           │
   │ │     7. Extended attributes...    │           │  ilename   Row 1 of 10
_  │ │     8. Access contol list...     │           │
_  │ │     9. Directory default ACL...  │           │  ogers
_  │ │    10. File default ACL...       │           │  .
_  │ └──────────────────────────────────┘ 14:27:11  │  artr2
a  │ Last changed  . . . . : 2003-01-16 14:27:11     │  c63
_  │ Last accessed . . . . : 2002-11-13 18:58:58     │  om
_  │ Created . . . . . . . : 2001-07-13 10:56:15     │  arry
_  │ Link count  . . . . . : 14                      │  ane
_  │  F1=Help         F3=Exit          F4=Name       │  dapsrv
_  │  F7=Backward     F8=Forward      F12=Cancel     │  yslogd
   └─────────────────────────────────────────────────┘
```

Figure 6-51 Select Option 8 to create the access ACL

When you press Enter after specifying either option 8, 9, or 10, you now have access to modify, add, or delete the ACL entries for the access ACL, directory default ACL, and the file default ACL.

6.52 Create an access ACL



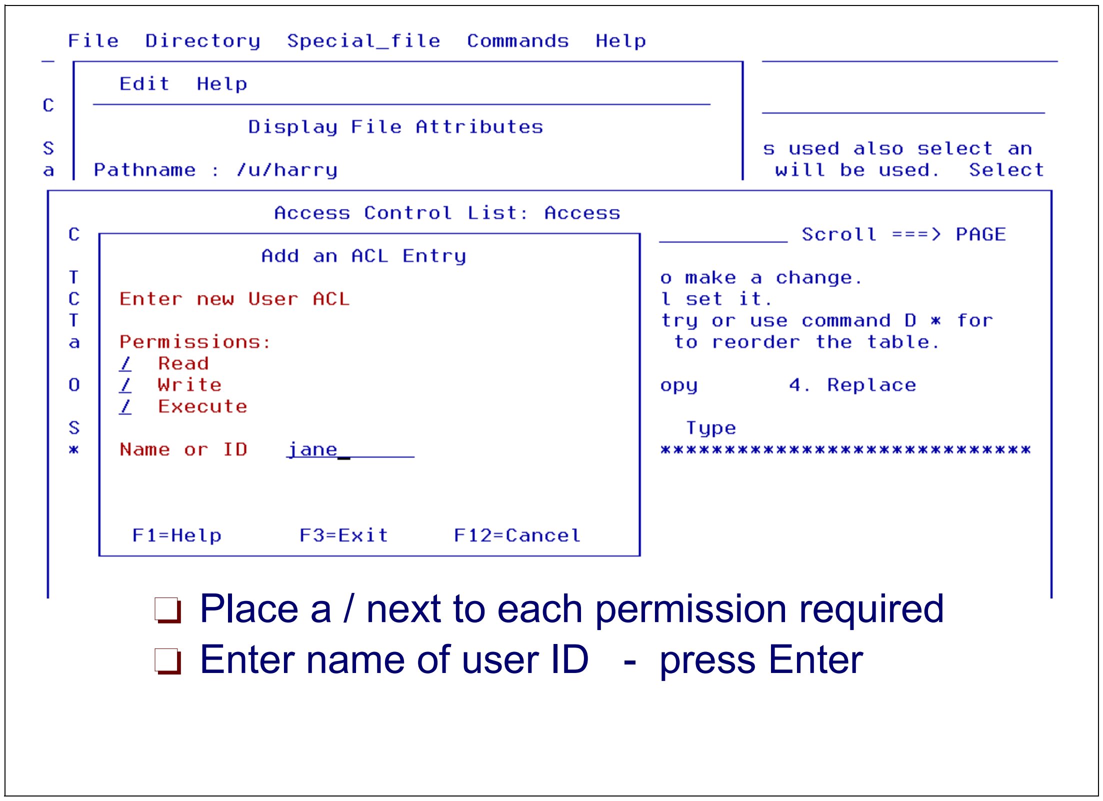
```
  File  Directory  Special_file  Commands  Help
_ ┌──────────────────────────────────────────────────┐ ______________________
  │  Edit  Help                                      │
C │ ________________________________________________ │ ______________________
  │                Display File Attributes           │
S │                                                  │ s used also select an
a │ Pathname : /u/harry                              │  will be used.  Select
┌─────────────────────────────────────────────────────────────────────────┐
│                  Access Control List: Access                            │
│ Command ===> _____________________________________________ Scroll ===> PAGE │
│                                                                         │
│ Type over read, write or execute permissions to make a change.          │
│ Clear the value to reset it, anything else will set it.                 │
│ To delete, place a D in the S column for an entry or use command D * for │
│ all entries. Use commands SORT ID or SORT NAME to reorder the table.    │
│                                                                         │
│ Option:     2  1. Add group  2. Add user   3. Copy      4. Replace      │
│                                                                         │
│ S          ID  Name      Read  Write  Execute  Type                     │
│ ***************************** Bottom of data ****************************** │
│                                                                         │
```

❑ Type a 2 on the Option line to create ACL - press Enter

Figure 6-52 Creating the access ACL for directory HARRY

This panel is used to create an ACL for a user ID to give access to a file or directory. In our example, we are going to give user JANE access to directory HARRY. To do this, place a 2 on the Option line and press Enter.

6.53 Add an access ACL

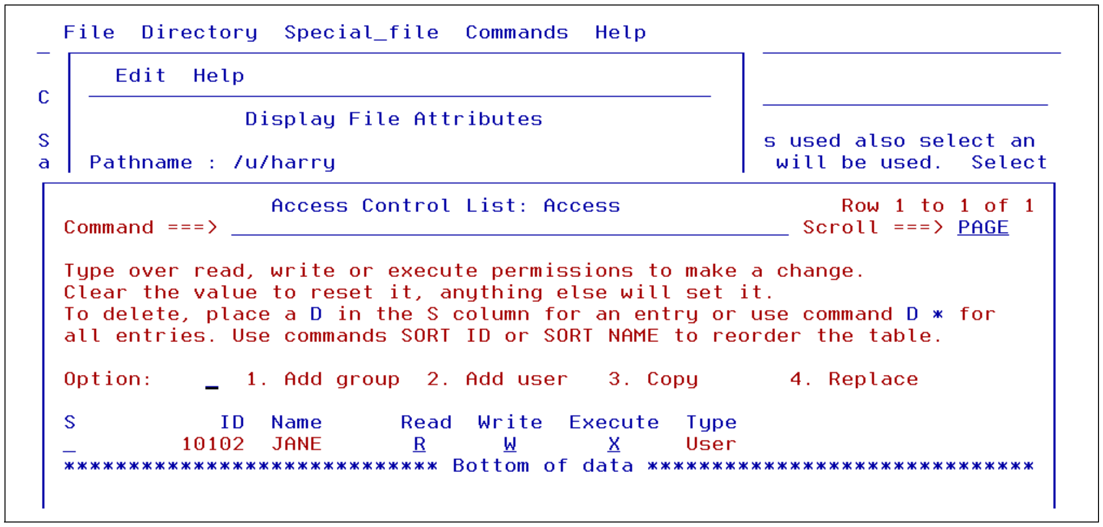


```
   File  Directory  Special_file  Commands  Help
 ─  ┌──────────────────────────────────────────────────┐  ──────────────────────
    │ Edit  Help                                        │
 C  │ ─────────────────────────────────────────────────  │  ─────────────────────
    │           Display File Attributes                 │
 S  │                                                   │  s used also select an
 a  │ Pathname : /u/harry                               │   will be used.  Select
  ┌─────────────────────────────────────────────────────────────────────────┐
  │             Access Control List: Access                                 │
  │ C ┌─────────────────────────────────────────┐ __________ Scroll ===> PAGE
  │   │          Add an ACL Entry               │
  │ T │                                         │ o make a change.
  │ C │ Enter new User ACL                      │ l set it.
  │ T │                                         │ try or use command D * for
  │ a │ Permissions:                            │  to reorder the table.
  │   │ /  Read                                 │
  │ O │ /  Write                                │ opy       4. Replace
  │   │ /  Execute                              │
  │ S │                                         │   Type
  │ * │ Name or ID   jane_______                │ *****************************
  │   │                                         │
  │   │                                         │
  │   │  F1=Help      F3=Exit      F12=Cancel   │
  │   └─────────────────────────────────────────┘
```

- Place a / next to each permission required
- Enter name of user ID - press Enter

Figure 6-53 The Add an ACL Entry window is used to create the ACL

Using this panel, you specify the permission bit settings that you require to give user JANE access to directory HARRY. In the example, we are giving user JANE `rwx` (read, write and execute) access to directory HARRY, as shown in the Pathname field in the figure. The following panel shows the newly created ACL after you press Enter.



```
   File  Directory  Special_file  Commands  Help
 ─  ┌──────────────────────────────────────────────────┐  ──────────────────────
    │ Edit  Help                                        │
 C  │ ─────────────────────────────────────────────────  │  ─────────────────────
    │           Display File Attributes                 │
 S  │                                                   │  s used also select an
 a  │ Pathname : /u/harry                               │   will be used.  Select
  ┌─────────────────────────────────────────────────────────────────────────────┐
  │ Command ===> ____________________________________  Scroll ===> PAGE         │
  │                Access Control List: Access                Row 1 to 1 of 1   │
  │                                                                             │
  │ Type over read, write or execute permissions to make a change.              │
  │ Clear the value to reset it, anything else will set it.                     │
  │ To delete, place a D in the S column for an entry or use command D * for    │
  │ all entries. Use commands SORT ID or SORT NAME to reorder the table.        │
  │                                                                             │
  │ Option:    _  1. Add group  2. Add user   3. Copy       4. Replace          │
  │                                                                             │
  │ S         ID   Name       Read  Write  Execute  Type                        │
  │ _      10102   JANE        R     W       X      User                        │
  │ ***************************** Bottom of data ******************************  │
```

6.54 ACL inheritance: New directory/new file

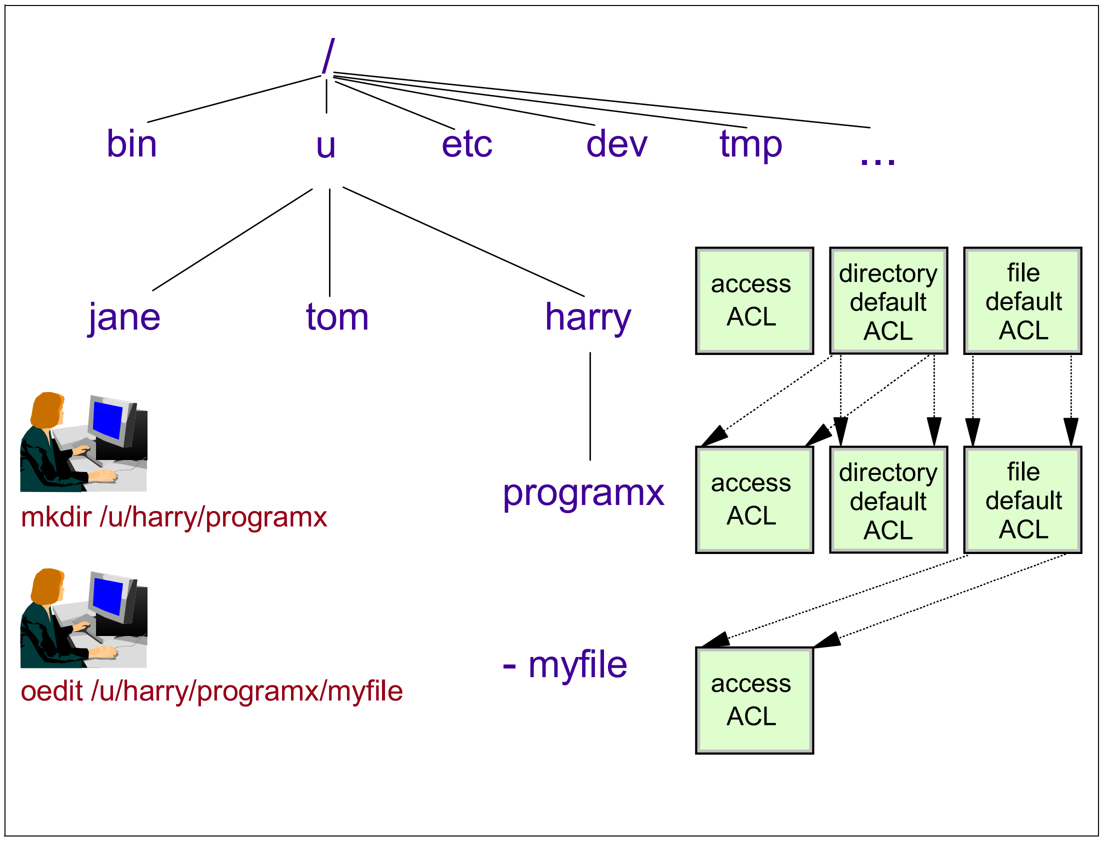


Figure 6-54 ACL inheritance

The directory structure is changed, as shown in Figure 6-54, by issuing the following command:

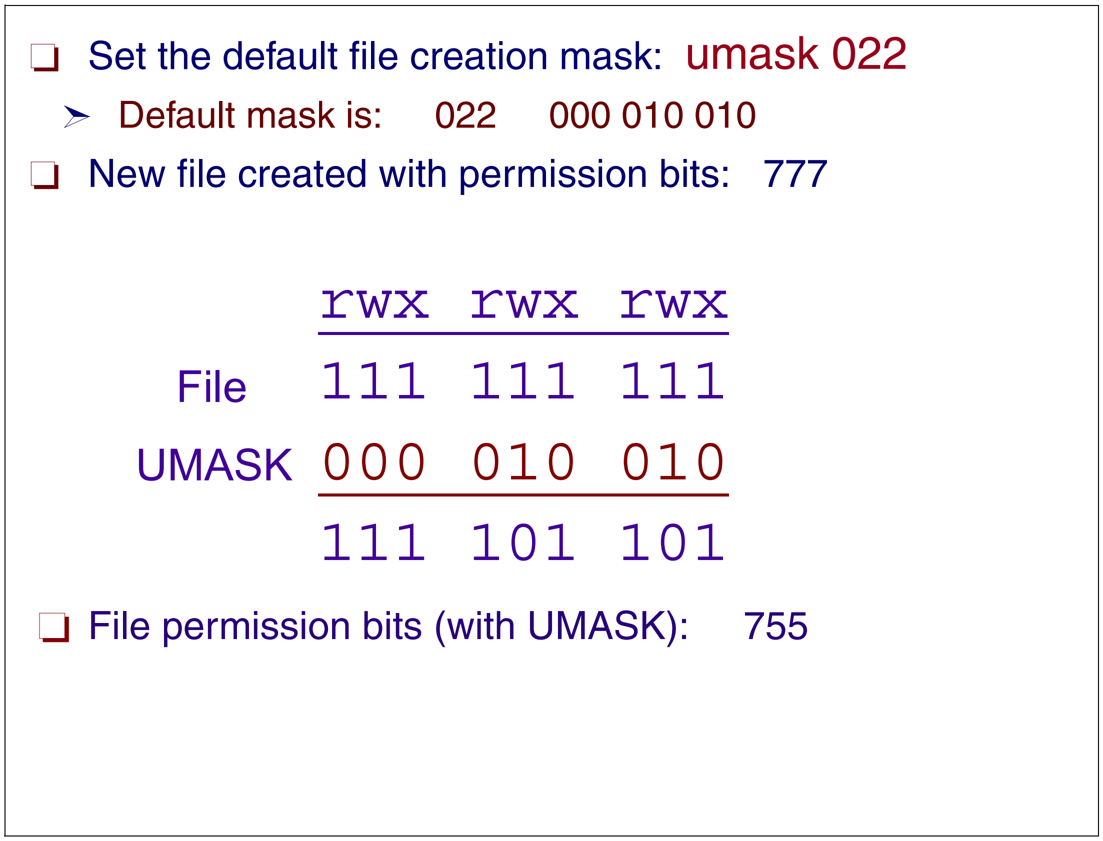
```
mkdir /u/harry/programx
```

The new directory, programx, inherits an access ACL from the directory default ACL of directory harry, and inherits the directory default ACL and file default ACL from directory harry.

ACL inheritance, as shown in the figure, associates an ACL with the newly created file, *myfile*, without requiring administrative action. However, it is not always (and in fact, may be seldom) necessary to apply ACLs on every file or directory within a subtree. If you have a requirement to grant access to an entire subtree (for example, a subtree specific to a given application), then access can be established at the top directory. If a given user or group does not have search access to the top directory, then no files within the subtree will be accessible, regardless of the permission bit settings or ACL contents associated with these files. The user or group will still need permission to the files within the directory subtree where appropriate. If this is already granted by the “group” or “other” bits, then no ACLs are necessary below the top directory.

Note: When defining ACLs, it is recommended to place ACLs on directories, rather than on each file in a directory.

6.55 Understanding UMASK



- ❑ Set the default file creation mask: umask 022
  - ➢ Default mask is: 022 000 010 010
- ❑ New file created with permission bits: 777

```
        rwx rwx rwx
File    111 111 111
UMASK   000 010 010
        111 101 101
```

- ❑ File permission bits (with UMASK): 755

Figure 6-55 Understanding the UMASK

When a file is created, it is assigned initial access permissions. If you want to control the permissions that a program can set when it creates a file or directory, you can set a file mode creation mask using the **umask** command.

The user can set this file mode creation mask for one shell session by entering the **umask** command interactively, or you can make the **umask** command part of your login. When you set the mask, you are setting a limit on allowable permissions: You are implicitly specifying which permissions are not to be set, even though the calling program may allow those permissions. When a file or directory is created, the permissions set by the program are adjusted by the umask value: the final permissions set a the program's permissions minus what the umask values restrict.

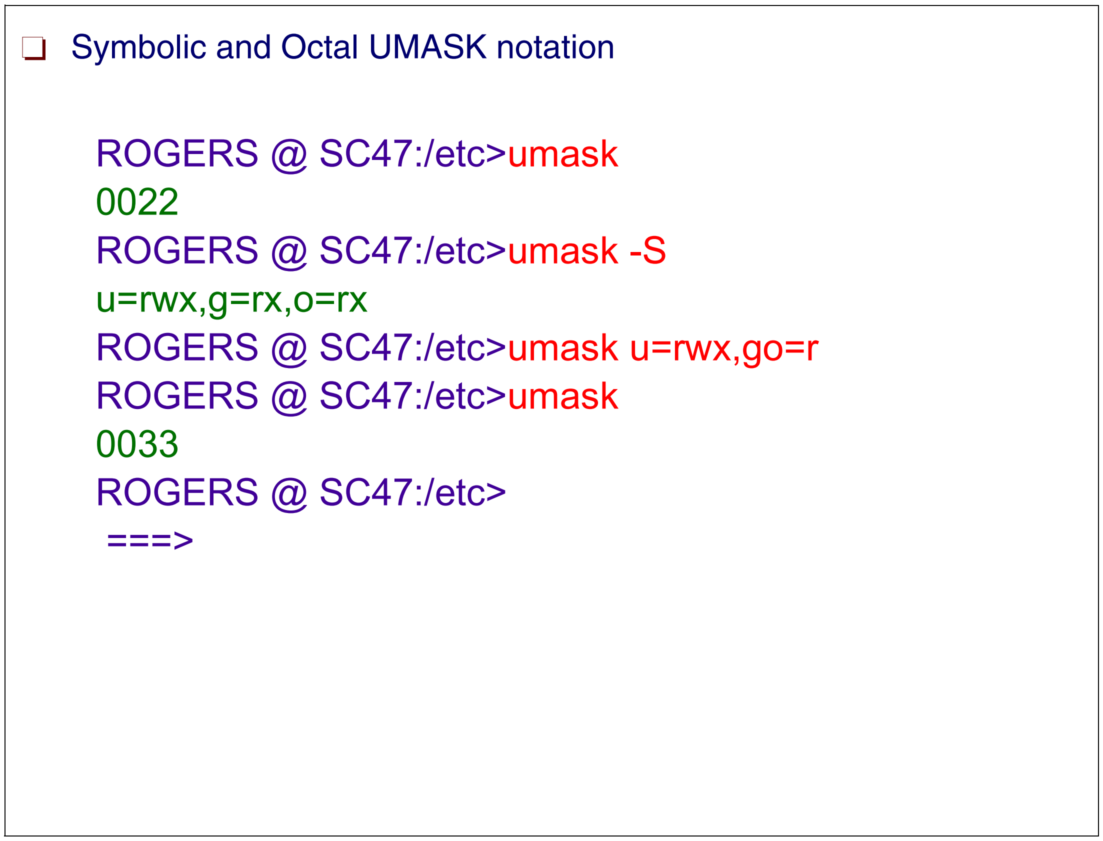
To use the **umask** command for a single session, enter:

```
umask mode
```

To create a mask that sets read-write-execute permission *on* for the owner of the file and *off* for everyone else, enter:

```
umask 077
```

6.56 Displaying the UMASK



❑ Symbolic and Octal UMASK notation

```
ROGERS @ SC47:/etc>umask
0022
ROGERS @ SC47:/etc>umask -S
u=rwx,g=rx,o=rx
ROGERS @ SC47:/etc>umask u=rwx,go=r
ROGERS @ SC47:/etc>umask
0033
ROGERS @ SC47:/etc>
 ===>
```

Figure 6-56 Displaying the UMASK

The umask sets the file-creation permission-code mask of the invoking process to the given mode.

The mode may be specified in symbolic (rwx) or octal format. The symbolic form specifies what permissions are allowed. The octal form specifies what permissions are disallowed.

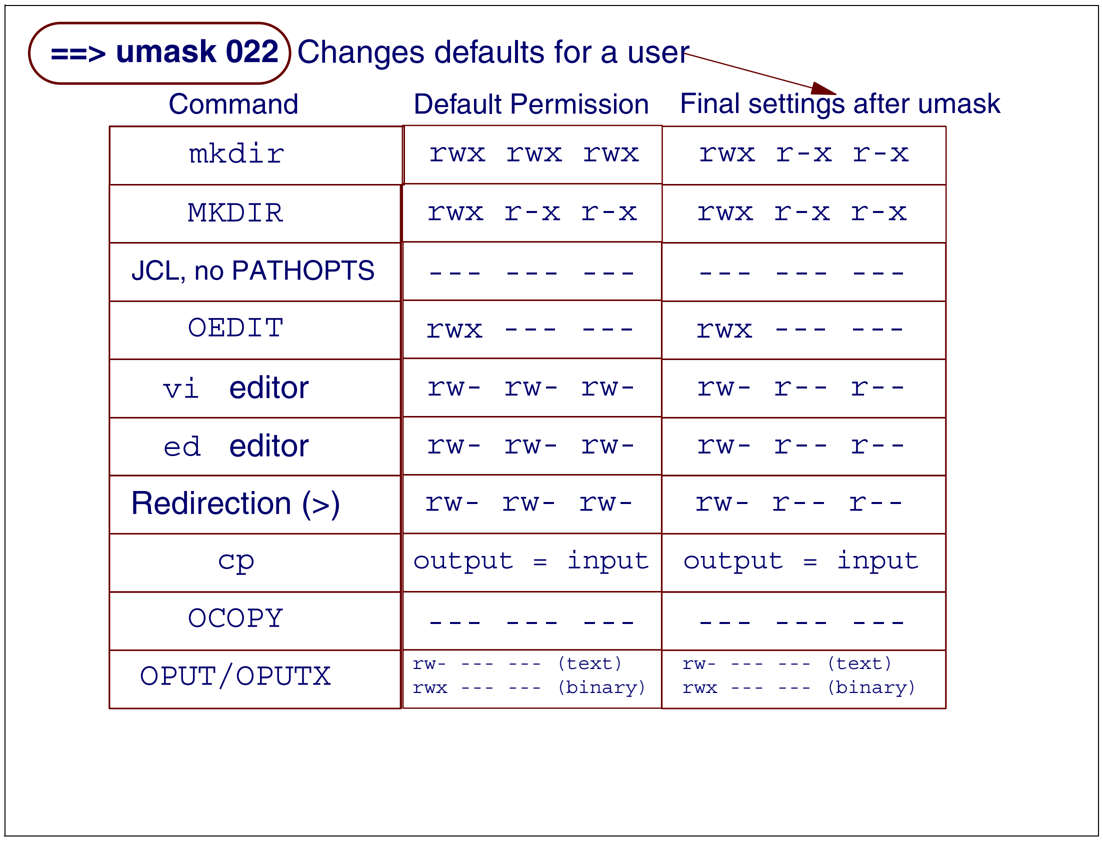
The file-creation permission-code mask (often called the umask) modifies the default (initial) permissions for any file created by the process. The umask specifies the permissions which are not to be allowed.

If the bit is turned off in the umask, a process can set it on when it creates a file. If you specify:

```
umask a=rx
```

you have allowed files to be created with read and execute access for all users. If you were to look at the mask, it would be 222. The write bit is set, because write is not allowed. If you want to permit created files to have read, write, and execute access, then set umask to 000. If you call **`umask`** without a mode argument, umask displays the current umask.

6.57 Default permissions and UMASK



==> umask 022 Changes defaults for a user

Command	Default Permission	Final settings after umask
mkdir	rwx rwx rwx	rwx r-x r-x
MKDIR	rwx r-x r-x	rwx r-x r-x
JCL, no PATHOPTS	--- --- ---	--- --- ---
OEDIT	rwx --- ---	rwx --- ---
vi editor	rw- rw- rw-	rw- r-- r--
ed editor	rw- rw- rw-	rw- r-- r--
Redirection (>)	rw- rw- rw-	rw- r-- r--
cp	output = input	output = input
OCOPY	--- --- ---	--- --- ---
OPUT/OPUTX	rw- --- --- (text) rwx --- --- (binary)	rw- --- --- (text) rwx --- --- (binary)

Figure 6-57 Default permissions and the UMASK

The system assigns default permission bits for files and directories at creation time. The settings depend on the type of command or facility that is used, and in some cases, on the type of file that is created.

A user can change the default setting when a file is created by using the **umask** shell command. The values set by the **umask** command will last for the length of the user's session, or the command can be part of the user's login so that the user always has the same default permissions.

PATHDISP indicates how MVS should handle the file when the job step ends normally or abnormally. This performs the same function as the DISP parameter for a data set.

umask

In /etc/profile, umask sets the default file creation mask. Using a umask of 022 causes a file 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.

The umask service changes the process's file creation mask. This mask controls file permission bits that are set whenever the process creates a file. File permission bits that are turned on in the file creation mask are turned off in the file permission bits of files that are created by the process. For example, if a call to the open service, BPX1OPN, specifies a “mode” argument with file permission bits, the process's file creation mask affects that argument. Bits that are on in the mask are turned off in the mode argument, and therefore in the mode of the created file.

If you use the **MKDIR** TSO/E command to create a directory, the permission bits will be different than if you use the shell command **mkdir** to do the same thing.

Each user can influence these defaults for his shell session by using the **umask** command. **umask** sets the file-creation permission-code mask of the invoking process to the given mode. You can specify the mode in any of the formats recognized by **chmod**.

As stated previously, the file-creation permission-code mask determines the default permissions for any file created by the process. For example, a file created by the **vi** command has the permissions specified by the umask unless the **vi** command specifies explicit permissions itself.

Redirecting command output to a file

Commands entered at the command line typically use the three standard files (STDIN, STDOUT, and STDERR), but you can redirect the output for a command to a file you name. If you redirect output to a file that does not already exist, the system creates the file automatically.

6.58 Create a new file example

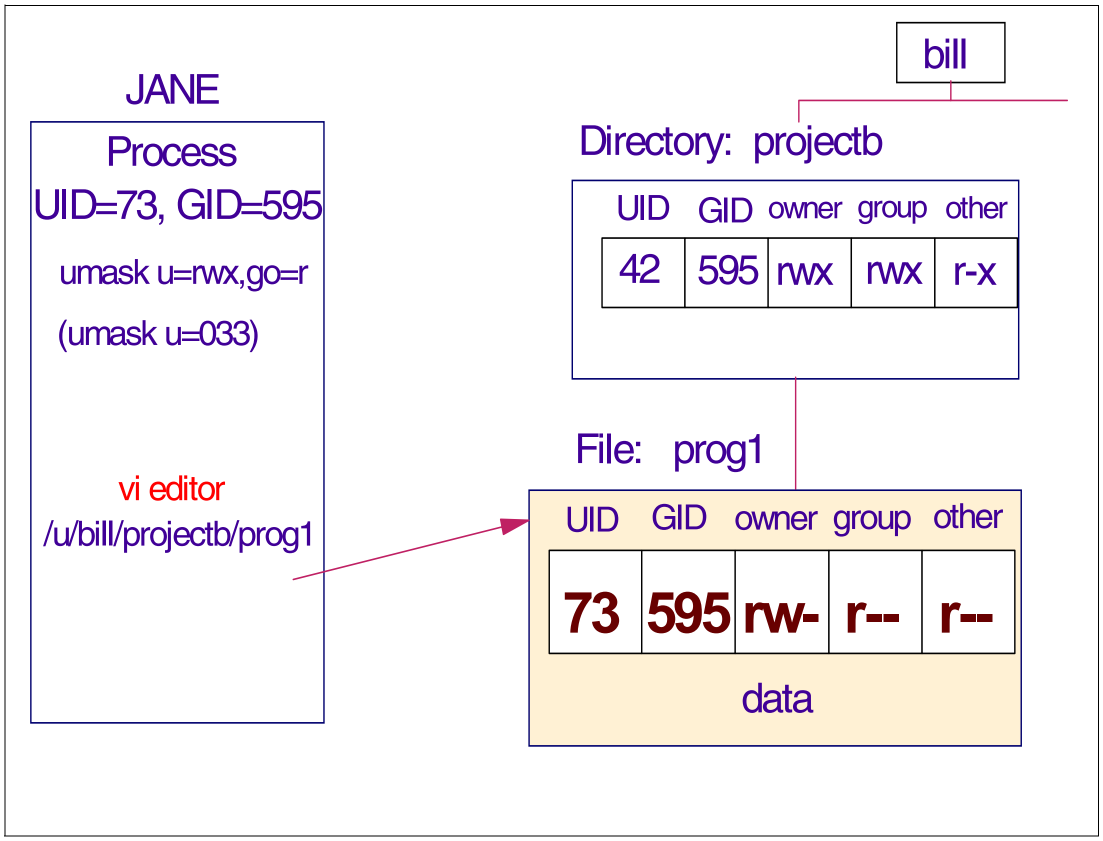


Figure 6-58 Example of creating a new file

In this example the user JANE (UID 73) is creating a new file. A file could be created in various ways. It could be copied from another file, it could be created with an editor, it could be moved, it could be created using JCL, and so on.

Security for a file is specified in the file security packet (FSP) which is a part of the attributes of the file. Each file has an FSP. The FSP is created when the file is created and is deleted when the file is deleted.

The contents of the FSP are determined as follows:

- The owning UID of the new file is taken from the effective UID (EUID) of the process that creates the file.
- The owning GID of the new file is taken from the owning GID of the directory for the new file.
- The file permission bits are set by the process that creates the file. The setting of the permission bits can be different, depending on the method used to create the file.

In this example, the user has set a umask to specify what the permissions should be for all files that the user creates. To do this, the user JANE issued the following shell command:

```
umask u=rwx,go=r
```

This will give the owner of the file `r`, `w`, and `x` access, while group and others will get `r` access.

In Figure 6-58, when the new file is created:

- The UID of the file is set to 73.
- The GID is set to 595.
- The file permission bits are set according to the umask as follows:
  - The file owner has `rw`, which mean read and write access.
  - Users in the group with a GID of 595 have `r--`, which means read access.
  - All other users (also known as world) have `r--`, which means read access.

In order to create a new file, the user must have write access to the directory for the new file.

6.59 Can user JOE access the file

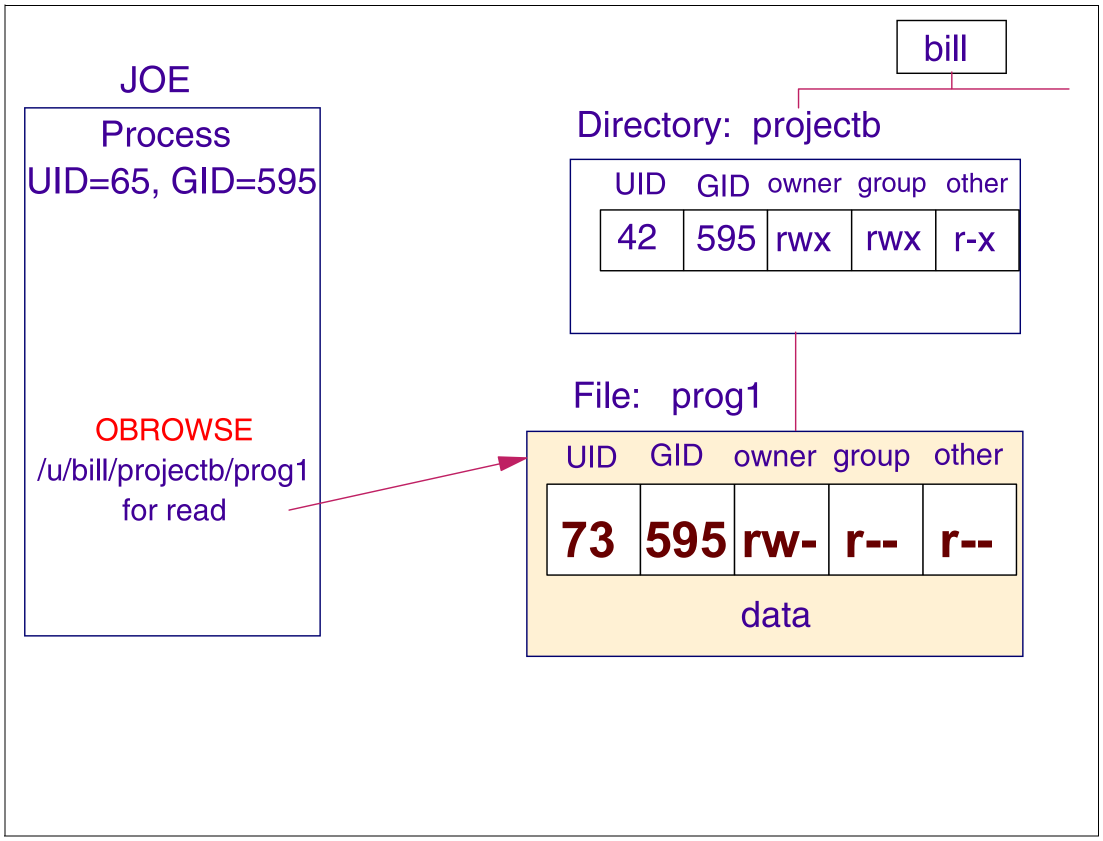


Figure 6-59 Can a user access the newly created file

In this example, another user who has a UID of 65 wants read access to the file /u/bill/projectb/prog1.

This user is not a superuser and is not a privileged or trusted started procedure.

The UID of the process, 65, does not match the file UID.

The GID of the process does match the file GID, and therefore the group permissions apply. The user wants read access and the permissions are `r--`, so the user is allowed access.

If the effective GID of the process did not match the file GID, RACF would have looked at the user's list of groups (assuming that RACF list of groups is active) for a group that has a GID that is the same as the file GID.

In order to access a file, the user must also have read and search permission to all directories in the path.

6.60 Can user ANN copy the file

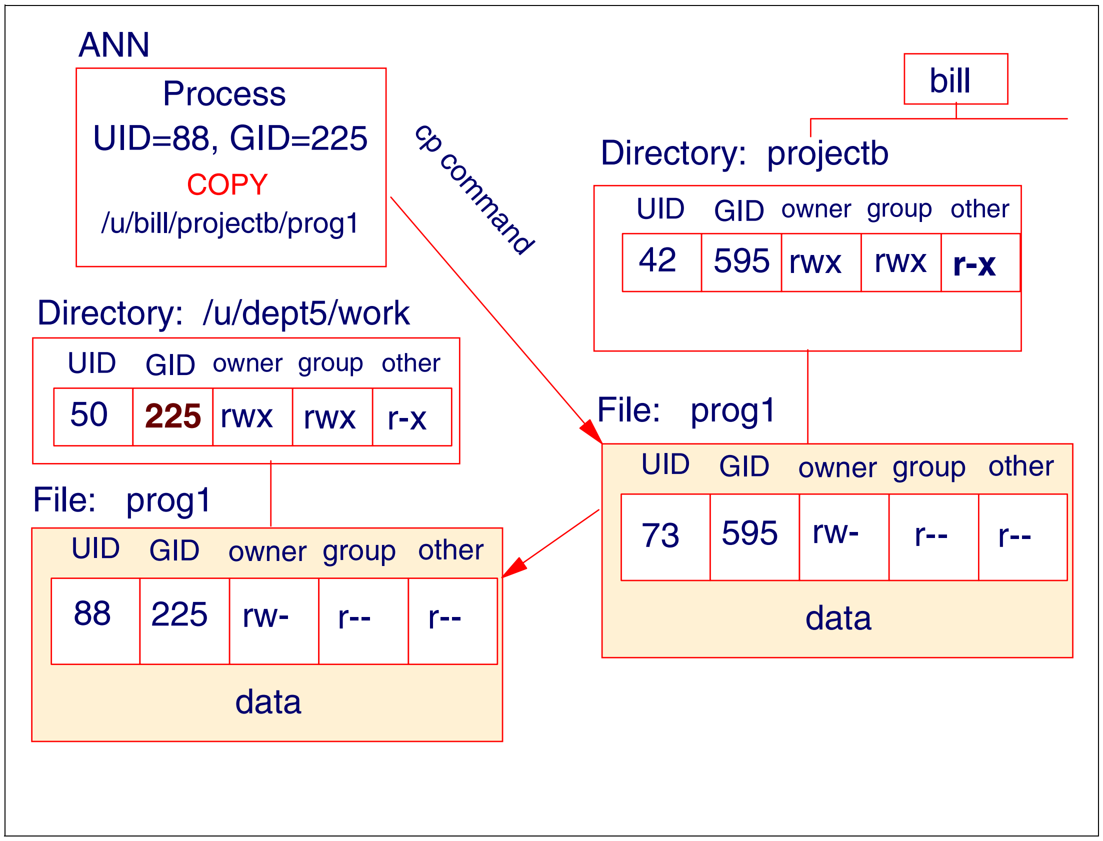


Figure 6-60 Can a user copy the newly created file

In this example, a user called ANN with a UID=88 wants to copy the file that user JANE created. The user is not a superuser, and is not a privileged or trusted started proc. This user is not the owner of the file and none of this user's groups have a GID that match the GID of the file.

In this case, access will be determined by the permissions for other users. Since the permission for other users is read access (`r--`), the user is allowed to copy the file.

User ANN does not have a umask set and the file permissions will be determined by the defaults for the **`cp`** command which was the method used for the copy. The new file will get the following attributes:

- The UID will be set to the UID of the person who copied the file. This will be 88.
- The GID will be GID of the directory where the file will be placed, which is 225.
- The file permission bits will be copied from the file *prog1* since this is the default for the **`cp`** command.

6.61 Setting file permissions

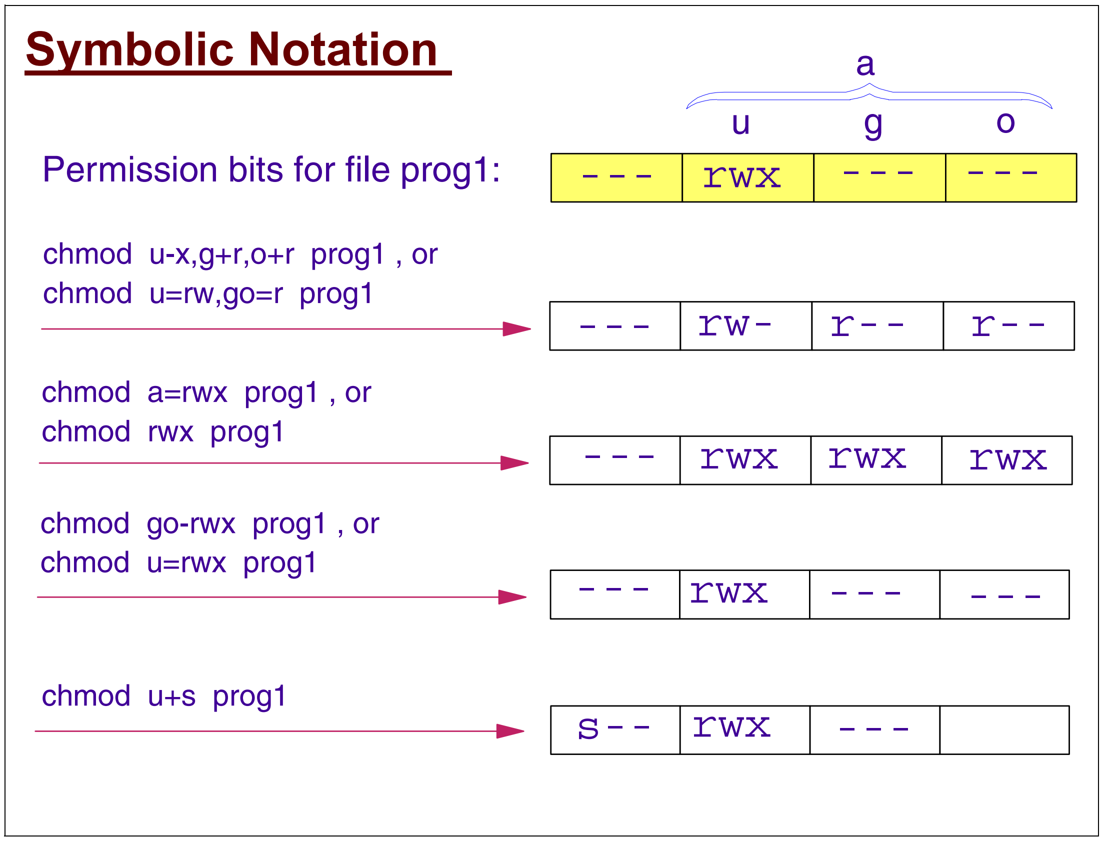
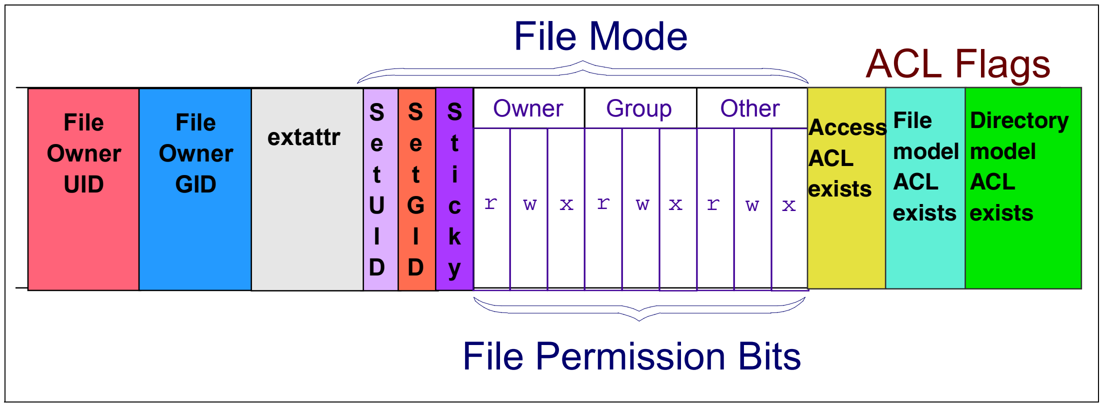


Figure 6-61 Commands to set permission bits

A file's mode mask includes the file permission bits, the SetUID bit, the SetGID bit, and the sticky bit. The file security packet (FSP) has the setuid, setgid, and sticky bits.



The initial permission bit for the file prog1 is shown in the box at the top right. The **chmod** command is used to make a change to the file mode mask of a file or directory, as follows:

- The z/OS UNIX shell command **chmod u-x,g+r,o+r** deletes execute (x) from the owner (u for user) permissions, adds read (r) to the group (g) permissions, and adds read (r) to the other (o) permissions.

- The same effect can be achieved with **`chmod u=rw,go=r`** which sets the owner (`u`) mask to read/write (`rw`), and sets the group and other (`go`) mask to read (`r`). When the equal sign is used, it turns on the bits specified and turns off all other bits.
- The command **`chmod a=rwx`** sets on the read, write, and execute bits for all (`a`) users, which includes the owner, group, and other.
- An equivalent command is **`chmod rwx`** in which the `a` (all users) is implied.
- In the command **`chmod go-rwx`**, `rwx` is turned off for group and other.
- An alternative form **`chmod u=rwx`** sets `rwx` on for the owner (u) mask, and turns off all other bits.
- The command **`chmod u+s`** shows how to turn on the SetUID bit. The **`s`** stands for set, and the **`u`** stands for UID. If we wanted to turn on the SetGID bit, we would use **`chmod g+s`**. Turning on the sticky bit would be achieved using **`chmod +t`**.

We cannot use RACF commands or panels to set the permissions. We use the z/OS UNIX shell commands. This is the same as AIX and UNIX. An alternative to the **`chmod`** command is to use the ISHELL menus. They may be more user-friendly for people who are not familiar with UNIX, and they provide help information.

Note: **`chmod`** can only be used by the file owner or a superuser.

6.62 Setting file permissions (2)

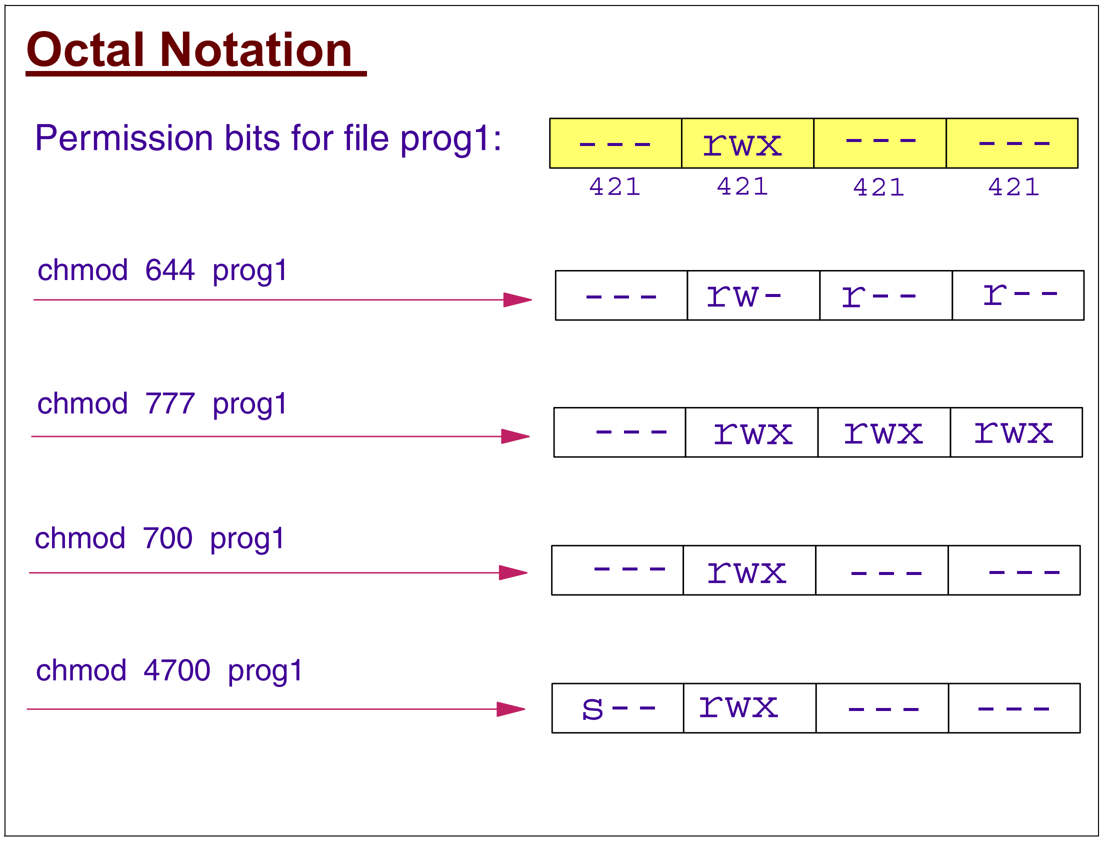


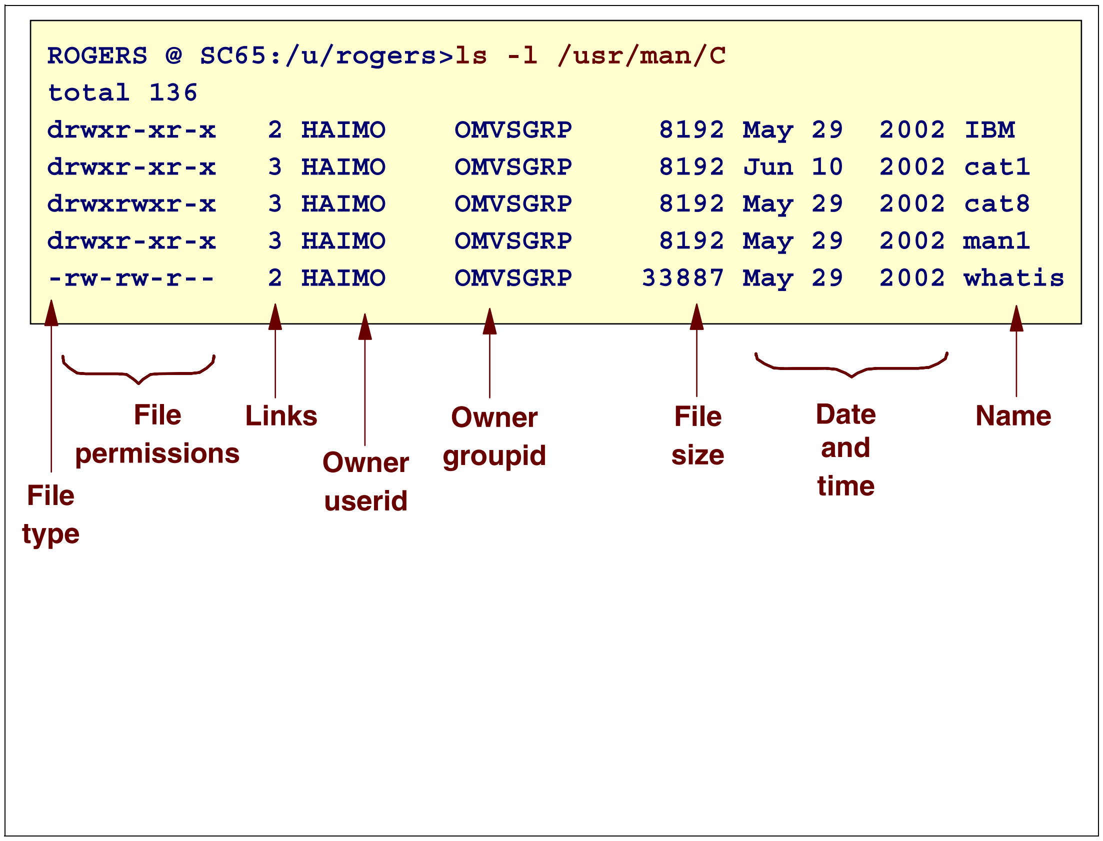
Figure 6-62 Commands to set the permission bits

Octal notation can be used on the **`chmod`** command instead of the symbolic notation. With octal notation, each set of three bits is represented in a single octal digit. For example, a permission of rwx would be represented as the octal digit 7 which is the sum of the 4 for read (r), the 2 for write (w), and 1 for execute/search (x), as follows:

- In the command **`chmod 644`** the octal 6 sets read and write (4+2) for the file owner, and sets read (4) for group and other users.
- The command **`chmod 777`** sets on read/write/execute (4+2+1) for the owner, group, and other users.
- The command **`chmod 700`** sets on the read, write, and execute bits (4+2+1) for the owner, and gives no access to group and other users.
- In the last command **`chmod 4700`** we see how to set the set UID, set GID, and sticky bits. This is done by using four octal digits, where the first digit represents the set UID, set GID, and sticky bits. Here, SetUID is the left-most bit (4), SetGID is the middle bit (2), and the sticky bit is the right-most bit (1).

Note: To specify permissions for a file or directory, you use at least a three-digit octal number, omitting the digit in the first position. When you specify three digits instead of four, the first digit describes owner permissions, the second digit describes group permissions, and the third digit describes permissions for all others.

6.63 List file and directory information



```
ROGERS @ SC65:/u/rogers>ls -l /usr/man/C
total 136
drwxr-xr-x   2 HAIMO    OMVSGRP     8192 May 29  2002 IBM
drwxr-xr-x   3 HAIMO    OMVSGRP     8192 Jun 10  2002 cat1
drwxrwxr-x   3 HAIMO    OMVSGRP     8192 May 29  2002 cat8
drwxr-xr-x   3 HAIMO    OMVSGRP     8192 May 29  2002 man1
-rw-rw-r--   2 HAIMO    OMVSGRP    33887 May 29  2002 whatis
```

Figure 6-63 Listing the file and directory permission settings

The **ls** command can be used to list important information about files and directories. If the **ls** command is used for a directory, it will display this information for all the files in the directory as shown in Figure 6-63.

Although there are 25 different options for the **ls** command, we only show the output for one option, **-l**, which stands for long. The file type in position 1 in the output indicates the type of file that is displayed on this line:

-	Regular file
d	Directory
b	Block special file (not supported for z/OS UNIX)
c	Character special file (for example, device)
l	Symbolic link
e	External link
p	FIFO special file (also known as a named pipe)
s	Socket file type

Positions 2 through 10 represent the file permissions for the owner, group, and other users.

6.64 Introducing daemons

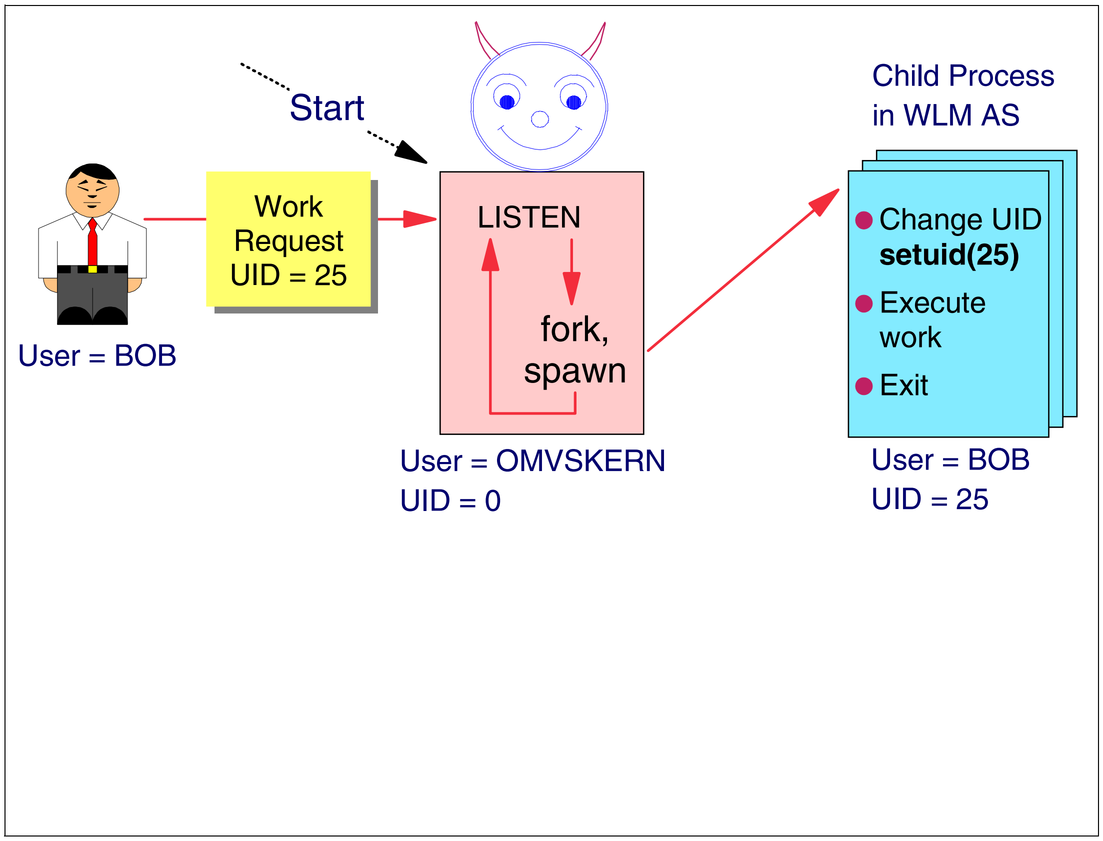


Figure 6-64 Introducing z/OS UNIX daemons

A daemon process is a process that runs in the background environment and is not associated with any particular terminal or user. Daemons run authorized (superuser authority) and can issue authorized functions like the following to change the identity of a user's process:

- setuid()
- seteuid()
- setreuid
- pthread_security_np()
- auth_check_resource_np()
- _login()
- _spawn() with user ID change
- _password()

Note: The spawn() service is allowed to create a new image under a specific user ID that is different from that of the invoker. When an invoker with appropriate privileges specifies a username on the _BPX_USERID environment variable or in the inheritance structure (INHEUSERID), the resulting image runs under the associated MVS user identity.

You can run daemons with regular UNIX security or with z/OS UNIX security. A z/OS UNIX daemon could also be described as a classical server process. Initially, the daemon process is started by an external command or event. Once started, the daemon *listens* for work requests from clients. When a request is received, the daemon will note the UID of the requester, and then fork a child process to carry out the request. Before executing the request, the daemon uses a special SYSCALL **`setuid`** to reset the security environment to match that of the requester.

For administrators, controlling daemons requires some extra considerations:

- How and when is a daemon process started (or restarted if it fails)?
- Daemons often need initialization options customized to installation requirements.
- Daemons have the ability to issue **`setuid()`**. Access to this type of power needs to be controlled, by controlling which programs can be daemons.
- A special user security profile BPXROOT must be created in order to support some daemon operations.

Note: The important point about the **`setuid`** instruction is that, in a z/OS environment, it will reset the whole security profile of the forked address space. The UID will be set to the requester's UID and the current RACF user ID information (the ACEE) will be changed to BOB to complement the UID. The requester's task therefore runs with access to both the UNIX and z/OS resources (data sets) owned by BOB.

Give daemon authority to the kernel. Most daemons that inherit their identities from the kernel address space are started from /etc/rc. To authorize the OMVSKERN user ID for the daemon FACILITY class profile, issue:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

In order for daemon processes to be able to invoke **`setuid()`** for superusers, define a superuser with a user ID of BPXROOT on all systems. To define the BPXROOT user ID, issue:

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/')
                PROGRAM('/bin/sh')  PASSWORD(A4B3C3D1)
```

Exactly how requests are passed to a daemon depends on the type of daemon. For TCP/IP-based daemons like the telnet or rlogin daemons, the user request will be passed over a socket connection from the IP network. For the cron daemon, the request is passed via a parameter area and a cross memory post to the daemon.

6.65 z/OS UNIX daemons

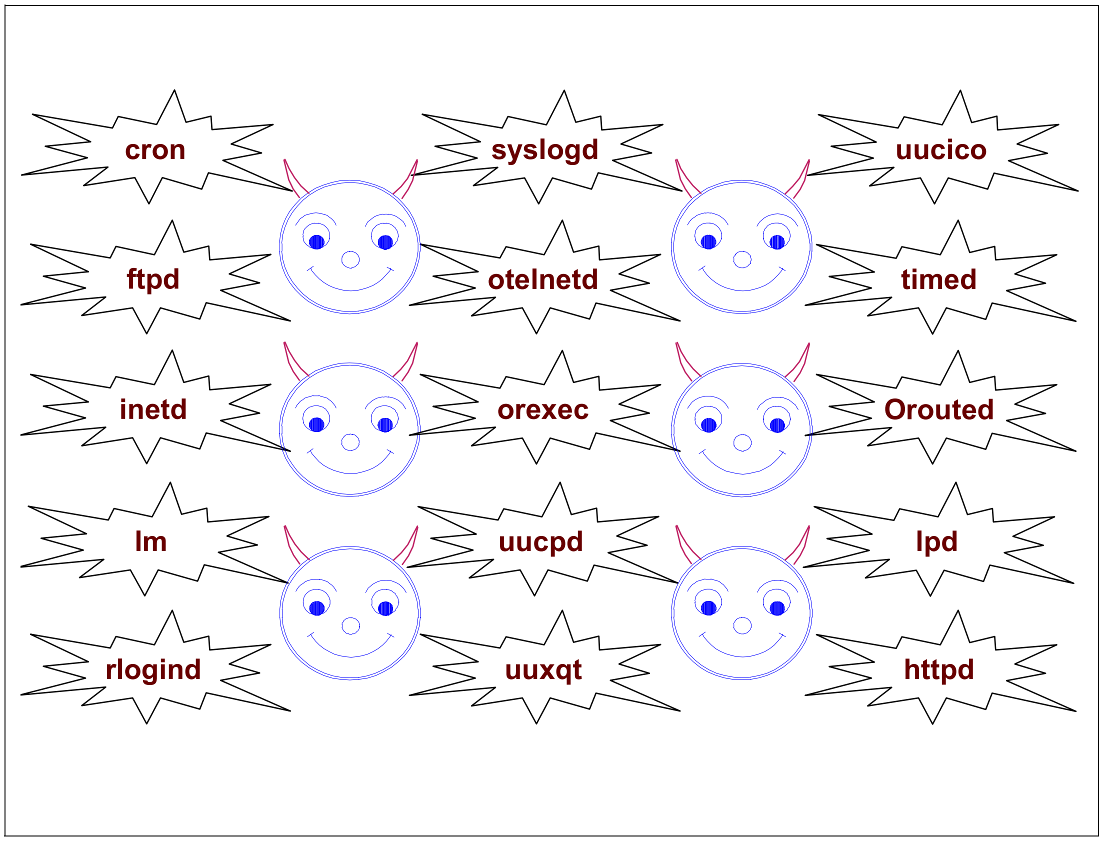


Figure 6-65 The z/OS UNIX daemons

A daemon is a long-lived process that runs unattended to perform continuous or periodic system-wide functions, such as network control. Some daemons are triggered automatically to perform their task; others operate periodically. Daemons typically encountered on z/OS UNIX systems include:

cron	The batch scheduler, cron is a clock daemon that runs commands at specified dates and times. You can specify regularly scheduled commands by using the **`crontab`** command. Jobs that are to be run only once can be submitted using the **`at`** or **`batch`** commands. cron runs commands with priorities and limits set by a queuedefs file.
ftpd	The file transfer daemon supplied with z/OS Communications Server.
inetd	The Internet daemon. The inetd daemon provides service management for a network. It starts the rlogind program or otelnetd program whenever there is either a remote login request or a remote telnet login from a workstation.
lm	The CS login monitor. The login monitor is a daemon that starts the login process in the shell for logins initiated by Communications Server (CS). CS nodes then forward service requests to the login monitor, which listens for the requests on a port number assigned to the lm service.
rlogind	The remote login daemon. The rlogind program is the server for the remote login command **`rlogin`**. It validates the remote login request and verifies the password of the target user. It starts a z/OS UNIX shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

syslogd	The syslog daemon supplied with z/OS Communications Server. Syslog daemon (syslogd) is a server process that has to be started as one of the first processes in your z/OS UNIX environment. Other servers and stack components use syslogd for logging purposes and can also send trace information to syslogd.
otelnetd	The remote logon daemon supplied with z/OS Communications Server.
orexecd	The remote execution protocol daemon supplied with z/OS Communications Server. The Remote Execution Protocol Daemon (REXECD) is the server for the REXEC routine. REXECD provides remote execution facilities with authentication based on user names and passwords.
uucpd	The UNIX-to-UNIX copy program daemon. The uucpd daemon is used to communicate with any UNIX system that is running a version of the UNIX-to-UNIX copy program. UUCP functions are used to automatically transfer files and requests for command execution from one UUCP system to another, usually in batch mode at particular times. Other daemons associated with uucp include:
uucico	Processes uucp and uux file transfer requests.
uuxqt	Runs commands from other systems.
Orouted	The Orouted daemon is supplied with z/OS Communications Server. The route daemon is a server that implements the Routing Information Protocol (RIP) (RFC 1058). It provides an alternative to the static TCP/IP gateway definitions.
lpd	The line printer daemon supplied with z/OS Communications Server. The line printer daemon enables printers from any TCP/IP host that is attached to the MVS spooling system.
timed	The time daemon supplied with z/OS Communications Server. The time daemon provides clients with UTC time. Network stations without a time chip obtain clocks from this daemon.
httpd	The http daemon supplied with WebSphere®.

6.66 UNIX-level security for daemons

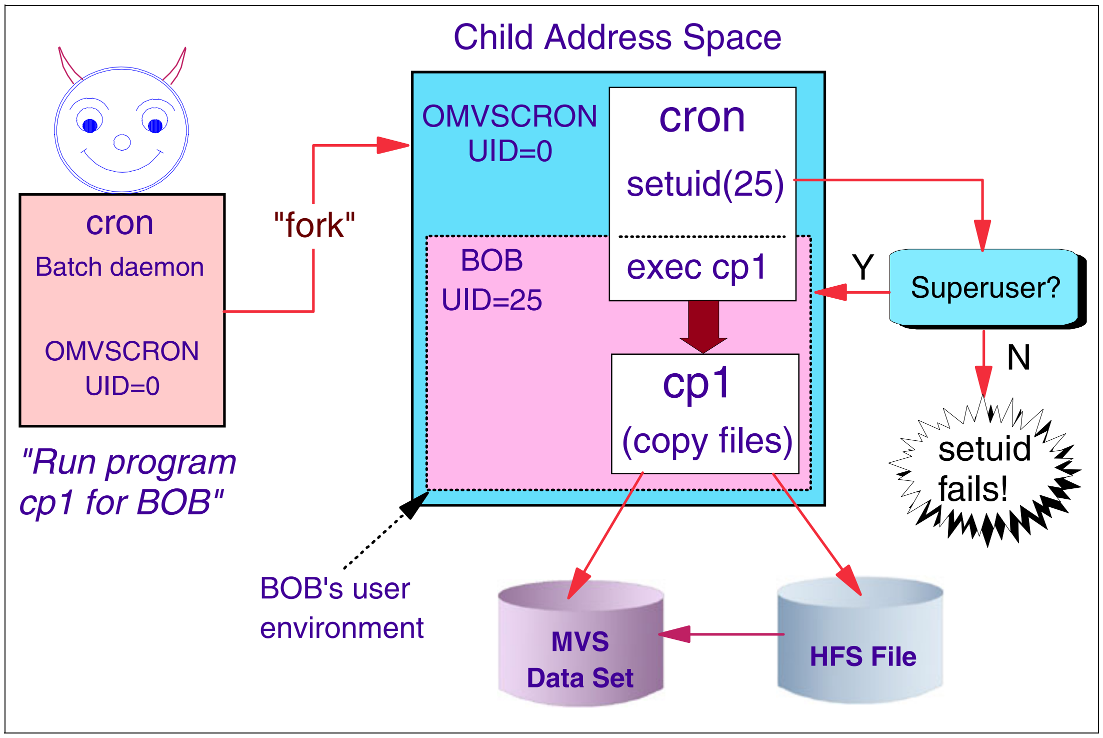


Figure 6-66 UNIX-level security for daemons

You can run daemons with UNIX-level security or with z/OS UNIX security. If the BPX.DAEMON (or BPX.SERVER) 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.

Daemons are processes that perform services for other users. In order to do this, a daemon must be able to change its identity temporarily to the identity of the user it will perform work for. The functions **`setuid()`** and **`seteuid()`** are authorized functions which will change the identity of a z/OS UNIX user or program.

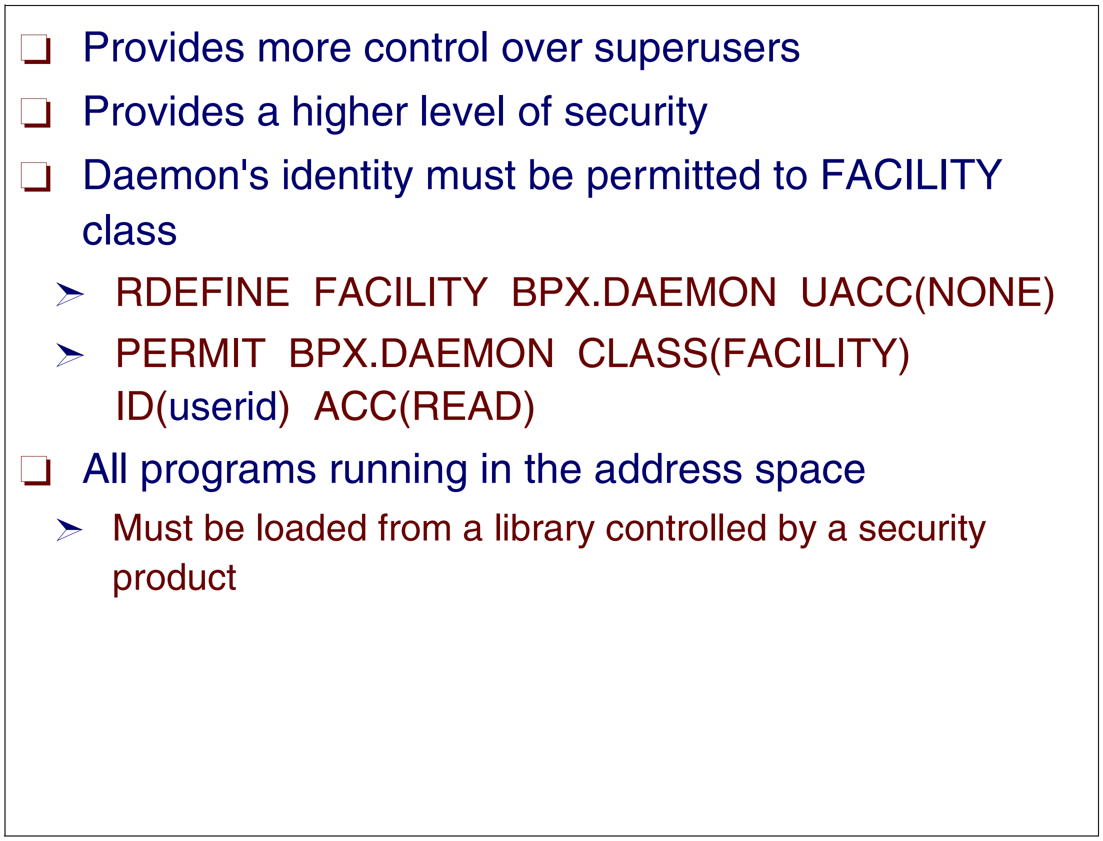
In fact, the daemon program clones itself using a **`fork()`** SYSCALL, and then the cloned child copy issues the **`setuid()`** request to initialize the security environment for the requester. Then the child issues **`exec()`** to pass control to the real request program to be executed. Superuser authority is required to use these functions, and this means that the daemon program (and its cloned child) must execute with superuser authority (UID=0).

UNIX-level security

For daemons, this means that all daemon programs must execute as superusers, and all superusers are allowed to use the **`setuid()`** and **`seteuid()`** functions to change the identity of a process to any other UID. Their z/OS identity will be changed to the one corresponding to the UID value; in the example, the cron daemon changes its identity to UID=25, which is the z/OS user ID BOB.

Note: If the target UID=0, and the target user ID is not known, the kernel sets default user ID=BPXROOT.

6.67 z/OS UNIX security: BPX.DAEMON



- ❑ Provides more control over superusers
- ❑ Provides a higher level of security
- ❑ Daemon's identity must be permitted to FACILITY class
  - ➢ RDEFINE FACILITY BPX.DAEMON UACC(NONE)
  - ➢ PERMIT BPX.DAEMON CLASS(FACILITY) ID(userid) ACC(READ)
- ❑ All programs running in the address space
  - ➢ Must be loaded from a library controlled by a security product

Figure 6-67 z/OS UNIX security with daemons

If the DAEMON (or BPX.SERVER) FACILITY class is defined, your system has z/OS UNIX security. Your system can exercise more control over your superusers.

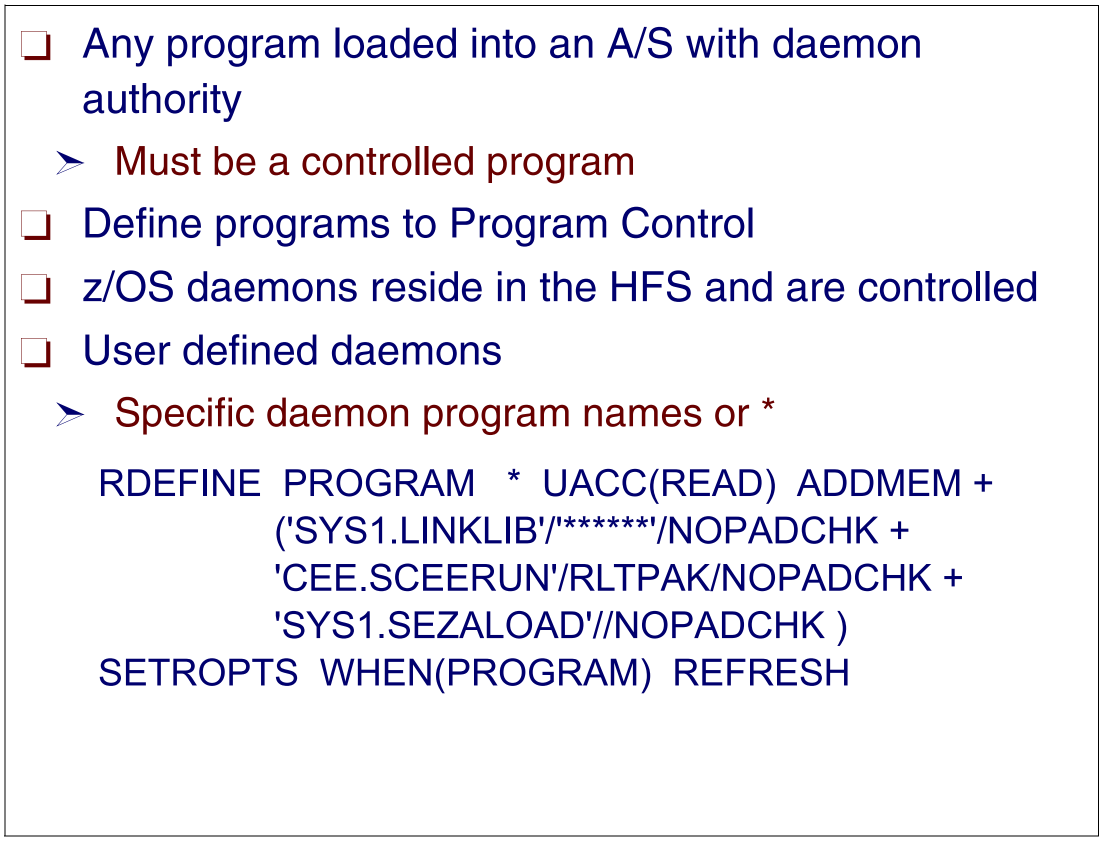
This level of security is for customers with very strict security requirements who need to have some superusers maintaining the file system but want to have greater control over the z/OS resources that these users can access. Although BPX.DAEMON provides some additional control over the capabilities 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. With BPX.DAEMON defined, a superuser process can successfully run these services if the following are true:

- The caller's user identity has been permitted to the BPX.DAEMON FACILITY class profile.
- All programs running in the address space have been loaded from a library controlled by a security product. A library identified to RACF Program Control is an example. Individual files in the HFS can be identified as controlled programs.

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.

6.68 RACF program control



- Any program loaded into an A/S with daemon authority
  - Must be a controlled program
- Define programs to Program Control
- z/OS daemons reside in the HFS and are controlled
- User defined daemons
  - Specific daemon program names or *

```
RDEFINE  PROGRAM   *  UACC(READ)  ADDMEM +
              ('SYS1.LINKLIB'/'******'/NOPADCHK +
              'CEE.SCEERUN'/RLTPAK/NOPADCHK +
              'SYS1.SEZALOAD'//NOPADCHK )
SETROPTS  WHEN(PROGRAM)  REFRESH
```

Figure 6-68 Defining RACF program control

The purpose of protecting load modules is to provide installations with the ability to control who can execute what programs and to treat those programs as assets.

You protect individual load modules (programs) by creating a profile for the program in the PROGRAM general resource class. A program protected profile in the PROGRAM class is called a controlled program.

The name of the profile can be complete, in which case the profile protects only one program, or the name of the profile can end with an asterisk (*), in which case the profile can protect more than one program. For example, a profile named ABC* protects all programs that begin with ABC, unless a more specific profile exists. In this way you can find which programs are causing the environment (such as PADS checking) to not work.

The profile for a controlled program must also include the name of the program library that contains the program and the volume serial number of the volume containing the program library. The profile can also contain a standard access list of users and groups and their associated access authorities.

Program access to data sets (PADS) allows an authorized user or group of users to access specified data sets with the user's authority to execute a certain program. That is, some users can access specified data sets at a specified access level only while executing a certain program (and the program access is restricted to controlled programs).

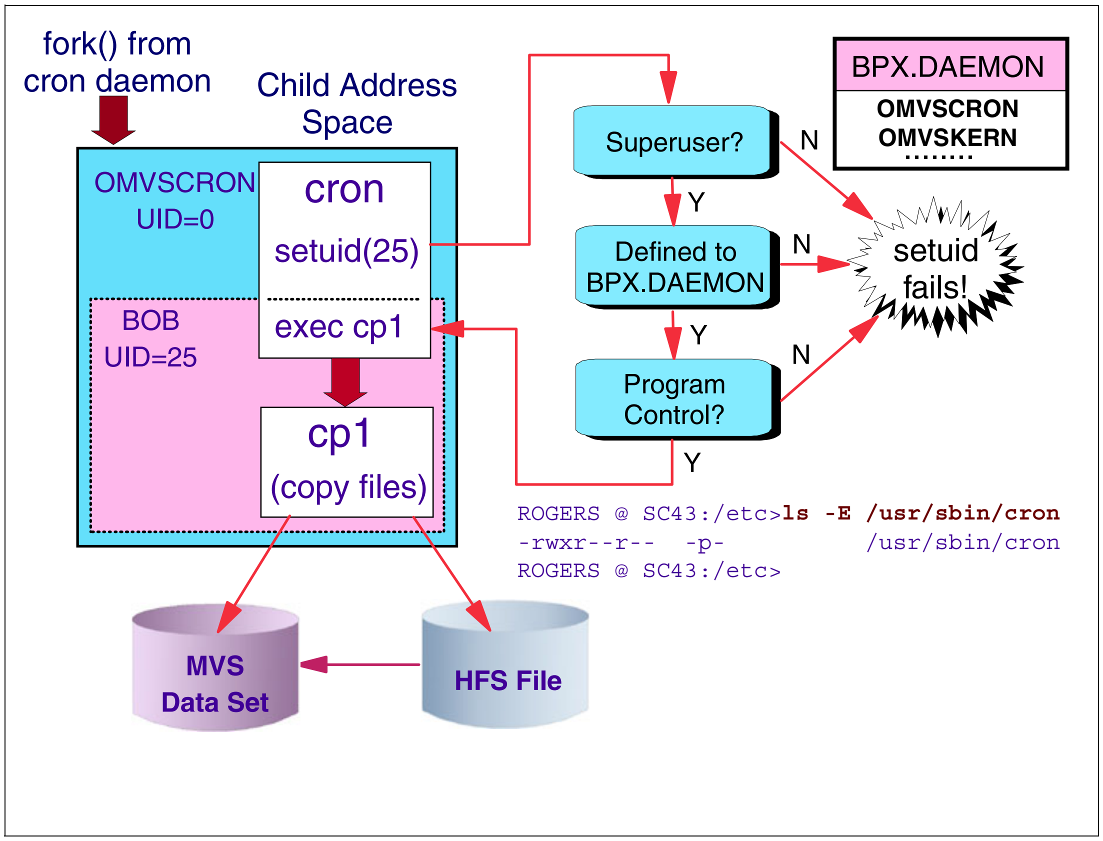
To set up program access to data sets, create a conditional access list for the data set profile protecting the data sets. To do this, specify `WHEN(PROGRAM(program-name))` with the `ID` and `ACCESS` operands on the `PERMIT` command. Specifying the `WHEN(PROGRAM)` operand requires that the user or group specified must be running the specified program to receive the specified access.

Choosing between the `PADCHK` and `NOPADCHK` Operands: With the `ADDMEM` operand of the **`RDEFINE`** and **`RALTER`** commands, you can also specify `PADCHK` or `NOPADCHK` as follows:

NOPADCHK `NOPADCHK` means that RACF does not perform the program-accessed data checks for the program. The program is loaded and has access to any currently opened program-accessed data sets, even though the user ID/program combination is not in the conditional access list. `NOPADCHK` allows an installation to define entire libraries of modules (such as the PL/I transient routines or ISPF) as controlled programs without having to give each of these modules explicit access to many program-accessed data sets. Use `NOPADCHK` if you trust the programs to access only data they should.

PADCHK `PADCHK` (the default) means that RACF checks for program-accessed data sets that are already open before executing the program. If there are any open program-accessed data sets, RACF ensures, before it allows this program to be loaded, that this user ID/program combination is in the conditional access list of each data set.

6.69 z/OS UNIX-level security for daemons



```
ROGERS @ SC43:/etc>ls -E /usr/sbin/cron
-rwxr--r--   -p-          /usr/sbin/cron
ROGERS @ SC43:/etc>
```

Figure 6-69 z/OS UNIX security with daemons

cron is a clock daemon that runs commands at specified dates and times.

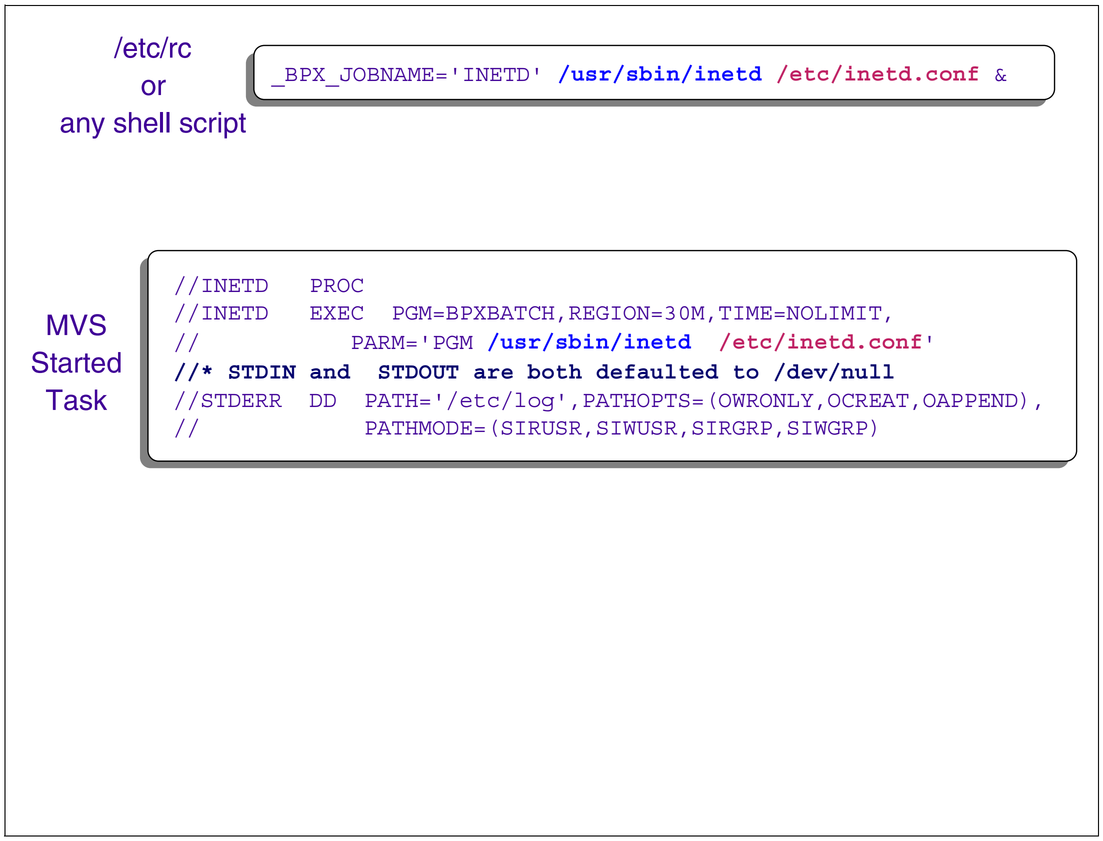
Figure 6-69 shows the cron daemon running a shell script for the user ID BOB (UID=25). The script will copy HFS files to an MVS data set. Before cron can run the script, it will fork a new process and set the identity of this process to UID=25 and the MVS identity to BOB. This will ensure that the script can be run successfully with BOB's shell environment and BOB's access to his MVS data sets. When the job is done, the cron child process will end, and cron will not have any access to BOB's MVS data sets.

To obtain a higher level of security in a z/OS system with daemons, the RACF FACILITY class called BPX.DAEMON can be used to control the use of the **setuid/seteuid** functions. Only the superusers permitted to the BPX.DAEMON class will be allowed to use the **setuid/seteuid** functions. In addition, there is a program control requirement.

When **setuid()** SYSCALL is issued, the caller (daemon) program, and all other programs currently loaded in the address space, must have been loaded from a z/OS data set with the RACF Program Control activated, meaning they must be controlled programs. As it is the cloned child daemon program that issues the request, it will inherit the contents of its address space from the parent daemon via fork.

This solution enables an installation to have some superusers which have authority to perform system maintenance (for example, to manage the hierarchical file system), while other special superusers (daemon user IDs) are allowed to change the identity of a process.

6.70 Start options for daemons


/etc/rc
or
any shell script
MVS
Started
Task

```
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &
```

```
//INETD   PROC
//INETD   EXEC  PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//             PARM='PGM /usr/sbin/inetd  /etc/inetd.conf'
//* STDIN and  STDOUT are both defaulted to /dev/null
//STDERR  DD   PATH='/etc/log',PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//             PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

Figure 6-70 Start options for daemons

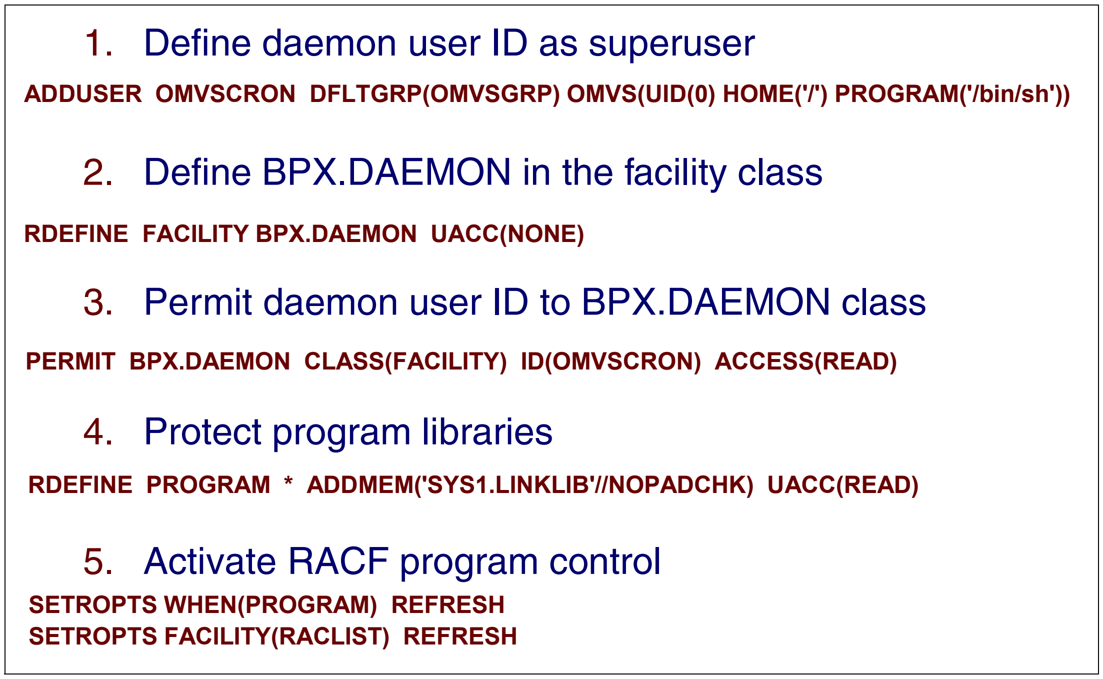
In a z/OS system, there are several ways of starting and restarting daemons. The method used depends on the level of control the installation has chosen for daemons. The daemon programs are installed in /usr/sbin. Daemons can be started using the following methods:

- To be started automatically when kernel is started, place the start options in the HFS file called /etc/rc. The initialization of z/OS UNIX includes running the commands in /etc/rc. The _BPX_JOBNAME environment variable assigns a job name to the daemon.
- As a cataloged procedure using the BPXBATCH program to invoke the daemon program.

If daemons need to be stopped, the **`kill`** command is typically used. Some daemons may have their own specific method of shutdown.

You should have appropriate procedures and directions in place to restart these daemons in case of failure. Started procedures are one way to do this; other methods may be more attractive depending on your automation strategy.

6.71 Define daemon security



1. Define daemon user ID as superuser

ADDUSER OMVSCRON DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))

2. Define BPX.DAEMON in the facility class

RDEFINE FACILITY BPX.DAEMON UACC(NONE)

3. Permit daemon user ID to BPX.DAEMON class

PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSCRON) ACCESS(READ)

4. Protect program libraries

RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'//NOPADCHK) UACC(READ)

5. Activate RACF program control

SETROPTS WHEN(PROGRAM) REFRESH
SETROPTS FACILITY(RACLIST) REFRESH

Figure 6-71 Defining daemon security

The following steps describe how to define security for a daemon:

1. Define a user ID for the daemon which is a superuser with UID=0, for example OMVSCRON:

   ```
   ADDUSER OMVSCRON DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
   ```

2. Define the BPX.DAEMON FACILITY class in RACF:

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

   Activate the class if this is the first RACF FACILITY class:

   ```
      SETROPTS CLASSACT(FACILITY) GENERIC(FACILITY) AUDIT(FACILITY)
      SETROPTS RACLIST(FACILITY)
   ```

3. Permit the daemon user ID to the BPX.DAEMON class:

   ```
   PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSCRON) ACCESS(READ)
   ```

4. Protect the program libraries that need to be protected from unauthorized updates:

   ```
   ADDSD 'SYS1.LINKLIB' UACC(READ)
   ADDSD 'SYS1.SCEERUN' UACC(READ)
   ADDSD 'SYS1.SEZALOAD' UACC(READ)
   ADDSD 'SYS1.SEZATCP' UACC(READ)
   ```

   Mark the data sets as controlled libraries:

   ```
      RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'//NOPADCHK +
                               'SYS1.SCEERUN'//NOPADCHK +
                               'SYS1.SEZALOAD'//NOPADCHK +
                               'SYS1.SEZATCP'//NOPADCHK) UACC(READ)
   ```

   Or, mark the daemon program as controlled instead of the whole library:

   ```
      RDEFINE PROGRAM CRON ADDMEM('SYS1.LINKLIB'//NOPADCHK)
      UACC(READ) AUDIT(ALL)
   ```

5. Activate RACF program control:

   Place the PROGRAM profile in storage:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

In many cases, a daemon program is started from the kernel and will inherit the kernel user ID, OMVSKERN. This example shows that it can have a separate user ID as long as the user ID is defined as a superuser. This superuser must be defined with a UID=0 in RACF, which means that this user cannot become a superuser by using the **su** command.

Note: This user ID should not have a TSO/E segment defined; only the OMVS segment is needed.

The name BPX.DAEMON must be used. No substitutions for this name are allowed. UACC(NONE) is recommended. If this is the first RACF FACILITY class defined in RACF, the SETROPTS command must be used to activate the class.

You can choose to protect a whole program library or individual load modules (members) in a library. The daemon program must reside in a program-controlled MVS partitioned data set, or in an HFS file with the extended attribute turned on via the **extattr +p** command. Both the program library for the daemons (for example, SYS1.LINKLIB) and the C runtime library must be protected.

The **ADDSD** command creates data set profiles for the data sets. You should protect against unauthorized updates so that nobody can replace a daemon program with a fake daemon program. If these profiles are already defined, this step can be skipped.

An installation has a choice of either protecting all programs in a program library or individual programs. To protect all members in a data set, specify **PROGRAM ***.

Note: Libraries in the LNKLIST concatenation are opened during IPL, and the programs in them are available to anyone unless the program name is defined as a controlled program.

When specifying program control for a program, you can choose to specify `UACC(READ)` or `UACC(NONE)`. `UACC(READ)` should be sufficient to protect the daemon program, for example:

6.72 Auditing options for z/OS UNIX

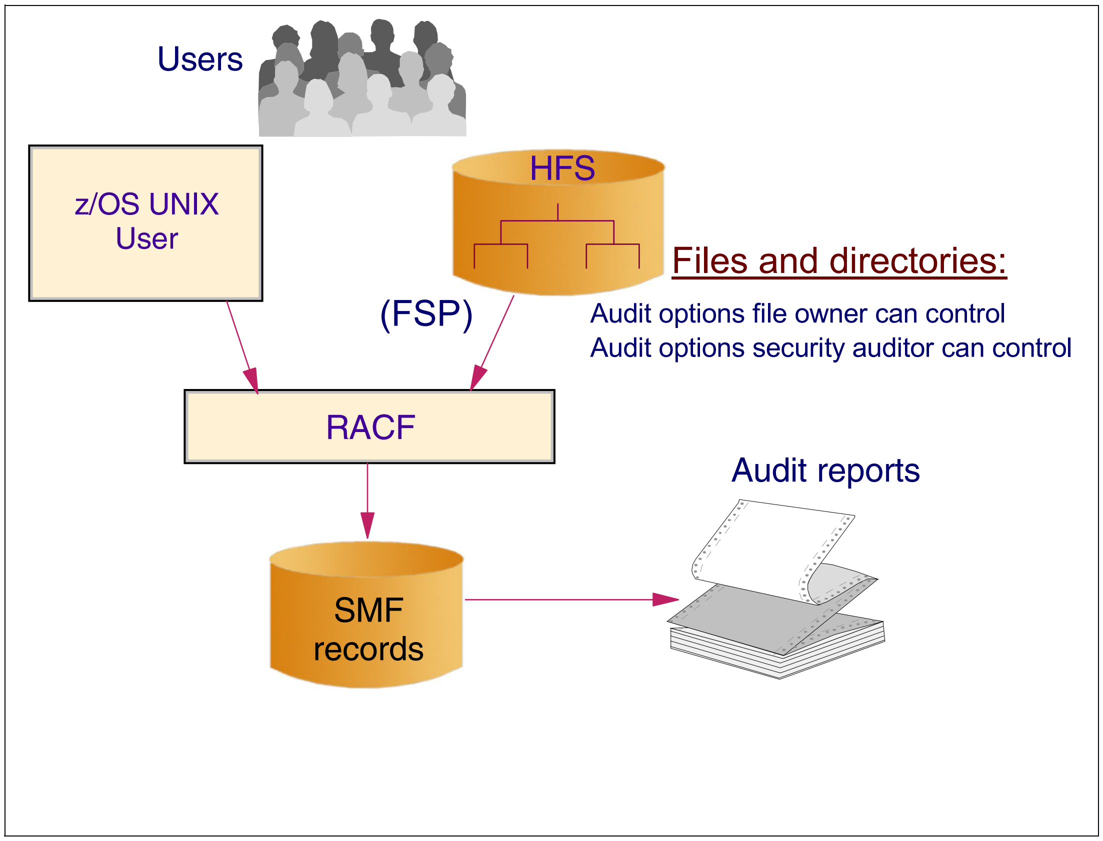

Users
z/OS UNIX User
HFS
Files and directories:
Audit options file owner can control
Audit options security auditor can control
(FSP)
RACF
Audit reports
SMF records

Figure 6-72 Auditing options for z/OS UNIX

Every file and directory has security information, which consists of:

- File access permissions
- UID and GID of the file
- Audit options that the file owner can control
- Audit options that the security auditor can control

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to z/OS UNIX resources. An SMF record can be written at each point where the system makes security decisions.

Six classes are used to control auditing of z/OS UNIX security events. These classes have no profiles. They do not have to be active to control auditing.

The security administrator or the file owners can also specify auditing at the file level in the file system.

RACF provides utilities for unloading SMF data (IRRADU00) and data from the RACF database (IRRDBU00), which can be used as input in audit reports.

6.73 File-based auditing

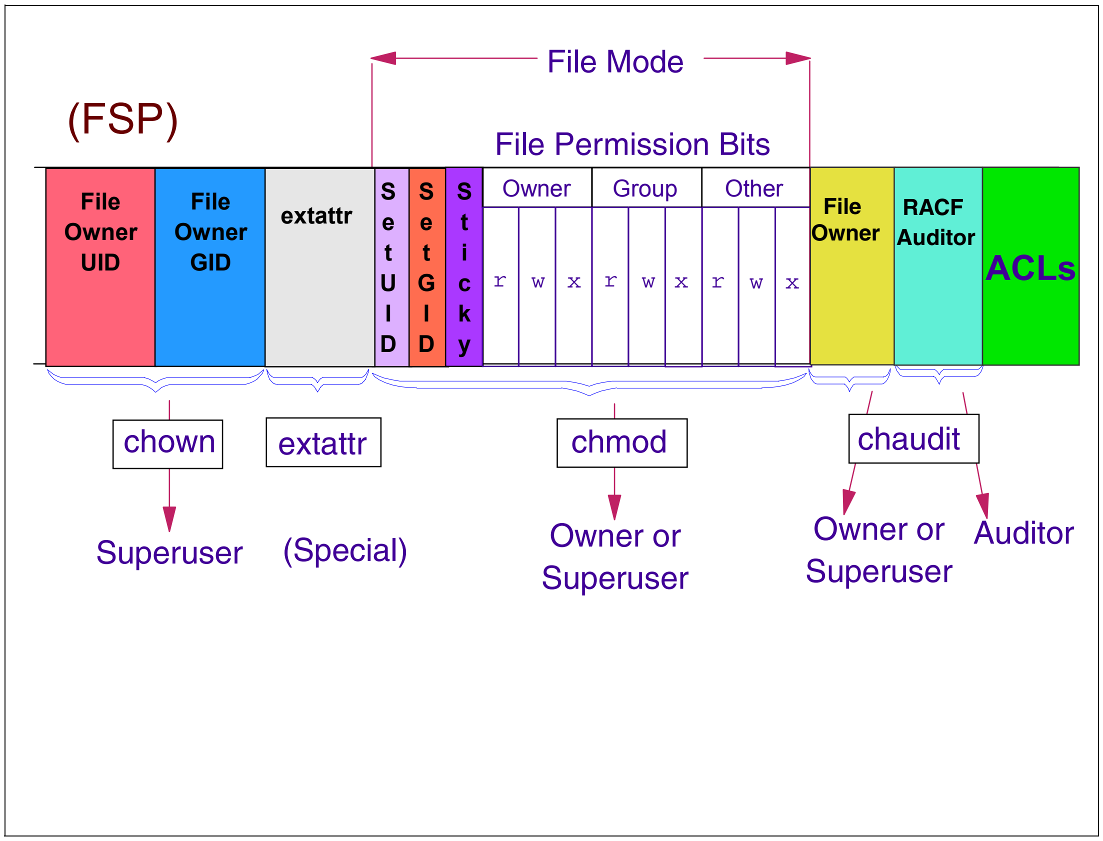


Figure 6-73 Auditing command for file access and the FSP

Auditing for file access is specified in the file security packet (FSP) with the **`chaudit`** command. Only a file owner or a security auditor can specify if auditing is turned on or off, and when audit records should be written for a directory or a file. There are two separate sets of auditing flags:

- Auditing set by the file owner (and superuser)
- Auditing set by the RACF AUDITOR

Audit records are written based on the combined owner and auditor settings. Auditing is set for read, write, and execute (search for directories) for the following kinds of accesses:

- Successful accesses
- Failures, that is, access violations
- All, which is both successes and failures
- None

When a file or a directory is created, default audit options are assigned. Different defaults are set for users and auditors. The same audit option is used no matter what kind of access is attempted (read, write, or execute). If auditing is not specified for a file, the defaults are:

- For owner auditing - audit failed accesses
- For RACF AUDITOR auditing - no auditing

When a file is created, these are the default audit options:

- User audit options: for all access types, audit_access_failed
- Auditor audit options: for all access types, don't_audit

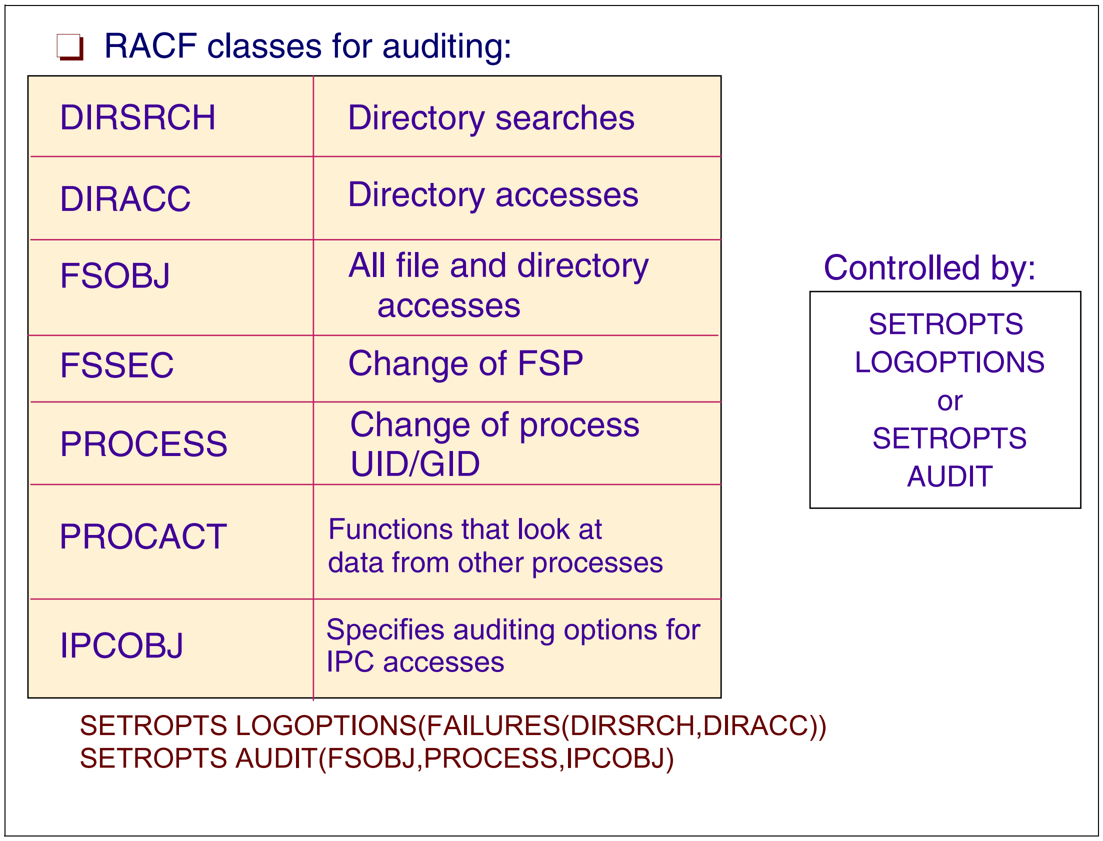
To change the audit options, you must use **`chaudit`**, a z/OS UNIX shell command, or the ISHELL menus. There are restrictions on who can change these options.

- For user audit options, you must be the owner of the file.
- For auditor audit options, you must have the RACF AUDITOR attribute. You can then change the auditor audit options for any file in the file system.

The default file-level audit options control the auditing of directory and file accesses. These defaults will only be used for a particular class (DIRSRCH, DIRACC, or FSOBJ) if **`SETROPTS LOGOPTIONS(DEFAULT(class))`** has been issued for that class.

The syntax of the **`chaudit`** command is similar to the syntax of the **`chmod`** command, which you use to set the security of the file. Use r,w,x to specify whether you want to audit reads, writes, and/or execute accesses.

6.74 Audit z/OS UNIX events



❑ RACF classes for auditing:

DIRSRCH	Directory searches
DIRACC	Directory accesses
FSOBJ	All file and directory accesses
FSSEC	Change of FSP
PROCESS	Change of process UID/GID
PROCACT	Functions that look at data from other processes
IPCOBJ	Specifies auditing options for IPC accesses

Controlled by:

SETROPTS LOGOPTIONS or SETROPTS AUDIT

```
SETROPTS LOGOPTIONS(FAILURES(DIRSRCH,DIRACC))
SETROPTS AUDIT(FSOBJ,PROCESS,IPCOBJ)
```

Figure 6-74 RACF classes used for auditing

RACF provides the following classes for auditing z/OS UNIX events:

DIRSRCH Controls auditing of directory searches.

DIRACC Controls auditing of access checks for read/write access to directories.

FSOBJ Controls auditing of all access checks for file system objects except directory searches via `SETROPTS LOGOPTIONS` and controls auditing of creation and deletion of file system objects via `SETROPTS AUDIT`.

FSSEC Controls auditing of changes to the security data (FSP) for file system objects.

PROCESS Controls auditing of changes to the UIDs and GIDs of processes and changing of the thread limit via the `SETROPTS LOGOPTIONS`, and controls auditing of dubbing, undubbing, and server registration of processes via `SETROPTS AUDIT`.

PROCAT Controls auditing of functions that look at data from or affect other processes.

IPCOBJ Specifies auditing options for IPC accesses and access checks for objects and changes to UIDs, GIDs, and modes. For access control and for z/OS UNIX user identifier (UID), z/OS UNIX group identifier (GID), and mode changes, use `SETROPTS LOGOPTIONS`. For object create and delete, use `SETROPTS AUDIT`.

Auditing can be controlled by using the commands **`SETROPTS LOGOPTIONS`**, and **`SETROPTS AUDIT`**, as follows:

- **`SETROPTS LOGOPTIONS(auditing_level(class_name))`** audits access attempts to the resources in the specified class according to the auditing level specified and can be used for all classes.
- **`SETROPTS AUDIT(class_name)`** specifies the names of the classes that RACF should audit. The AUDIT option can be used for the classes FSOBJ, IPCOBJ, and PROCESS.
- **`SETROPTS LOGOPTIONS(DEFAULT)`** indicates you want no class auditing and only file-level auditing (use **`chaudit`** to specify).

Audit records are always written when:

- A user who is not defined as a z/OS UNIX user tries to dub a process.
- A user who is not a superuser tries to mount or unmount a file system.
- A user tries to change a home directory.
- A user tries to remove a file, hard link, or directory.
- A user tries to rename a file, hard link, symbolic link, or directory.
- A user creates a hard link.

Note: There is no option to turn off these audit records.

The auditing levels for LOGOPTIONS are:

ALWAYS	All access attempts to resources protected by the class are audited.
NEVER	No access attempts to resources protected by the class are audited (all auditing is suppressed).
SUCCESSES	All successful access attempts to resources protected by the class are audited.
FAILURES	All failed access attempts to resources protected by the class are audited.
DEFAULT	Auditing is controlled by the auditing bits in the FSP for z/OS UNIX files and directories.

6.75 Chaudit command

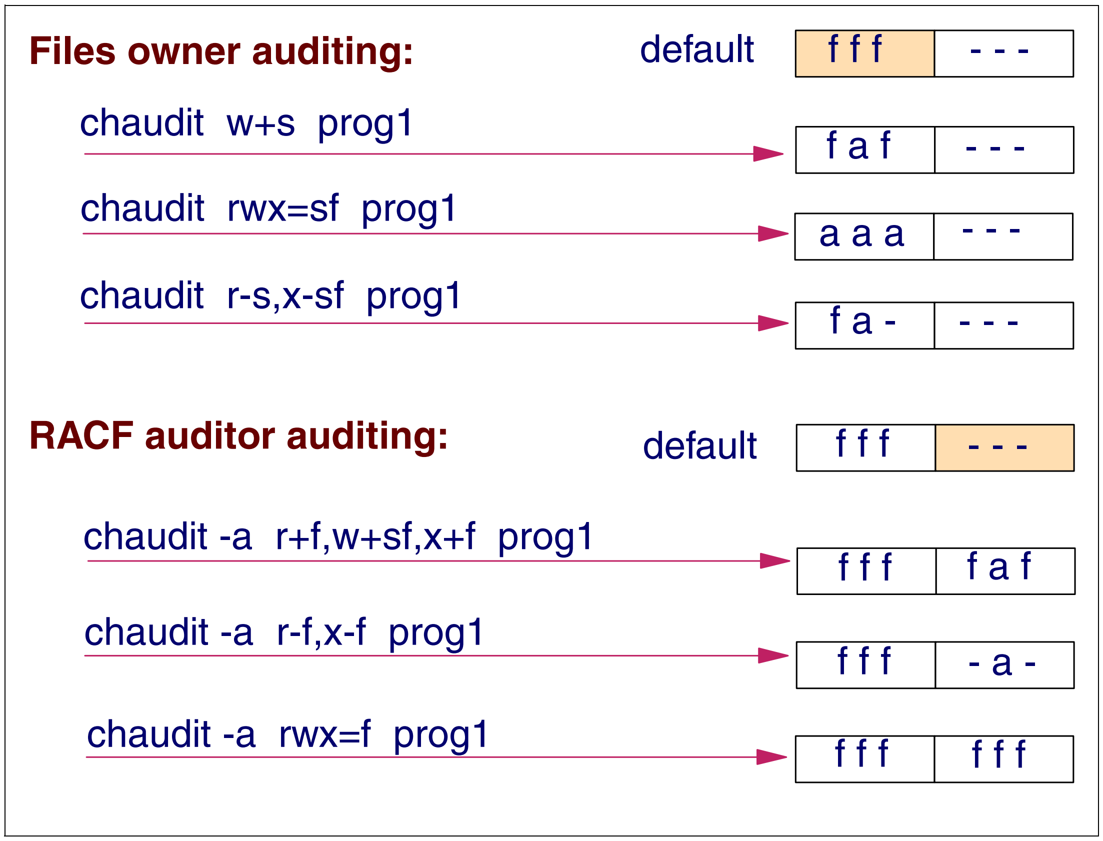


Figure 6-75 **`chaudit`** *command to set auditing options*

The **`chaudit`** command changes the audit attributes of the specified files or directories. Audit attributes determine whether or not accesses to a file are audited by the system authorization facility (SAF) interface.

Note: **`chaudit`** can be used only by the file owner or a superuser for non-auditor-requested audit attributes. It takes a user with auditor authority to change the auditor-requested audit attributes.

The top part of Figure 6-75 shows examples of the **`chaudit`** command usage by the file owner (or superuser). The default audit settings are shown in the upper right-hand corner of the figure, then the command is applied as follows:

- **`chaudit w+s prog1`** adds (`+`) auditing for successful accesses (`s`) for write accesses (`w`).
- **`chaudit rwx=sf prog1`** specifies that all (`a`) accesses, that is both successes (`s`) and failures (`f`), are to be audited for reads, writes, and executes.
- **`chaudit r-s,x-sf prog1`** says to stop (`-`) auditing successes (`s`) for read (`r`), and stop (`-`) auditing both successes (`s`) and failures (`f`) for execute (`x`) access. The same effect can be achieved with the command **`chaudit r=f,x= prog1`**.

Examples of the **`chaudit`** command usage by the RACF Auditor are shown in the lower part of Figure 6-75. The default audit settings shown first, then the command is applied as follows:

- **`chaudit -a r+f,w+s,x+f prog1`** adds auditing of successes (`s`) and failures (`f`) for write access, and specifies to write an audit record whenever an access failure (`f`) occurs for read or execute accesses.
- **`chaudit -a r-f,x-f prog1`** turns off (`-`) auditing for failures (`f`) for read and execute accesses.
- **`chaudit -a rwx=f prog1`** turns on auditing for unsuccessful (`f`) read, write, and execute accesses.

Note that the auditor includes the option **`-a`** when issuing the **`chaudit`** command, and that the auditor can only set the audit flags in the auditor's section of the FSP.

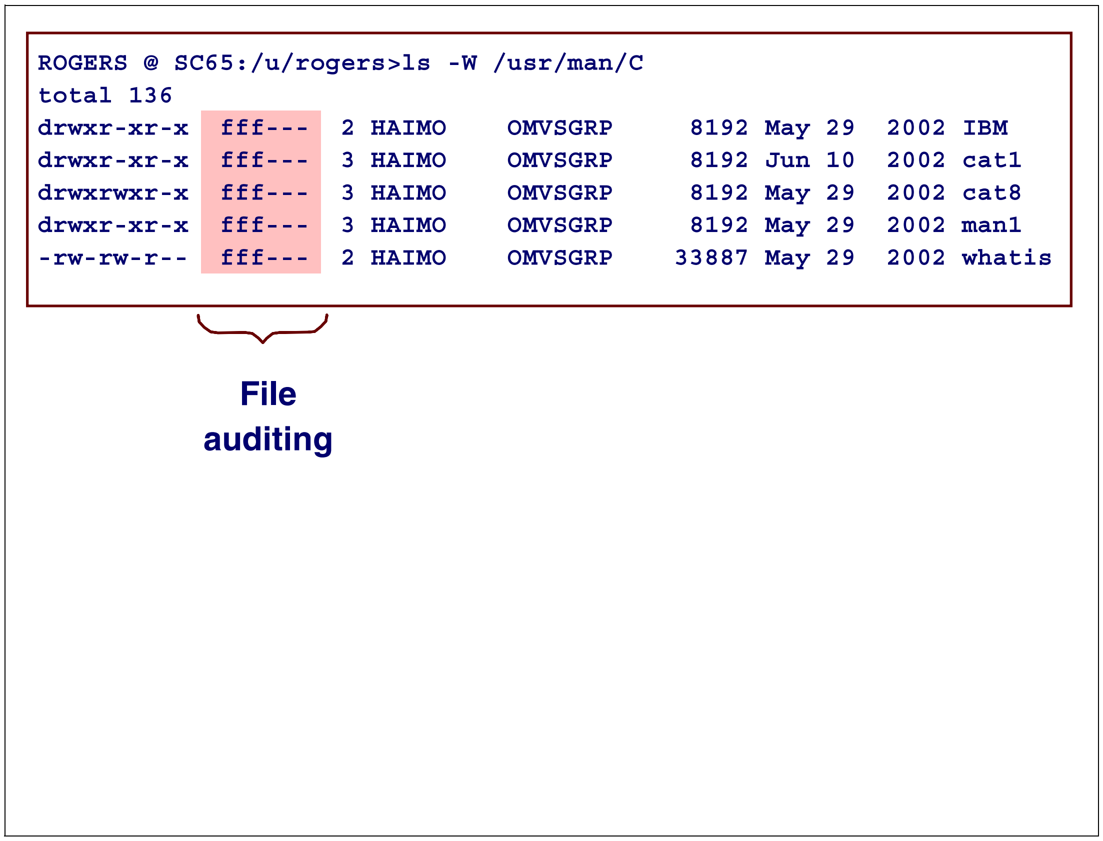
The audit condition part of a symbolic mode is any combination of the following:

s	Audit on successful access if the audit attribute is on.
f	Audit on failed access if the audit attribute is on.

The following command changes the file prog1 so that all successful and unsuccessful file accesses are audited:

```
chaudit rwx=sf prog1
```

6.76 List audit information for files



```
ROGERS @ SC65:/u/rogers>ls -W /usr/man/C
total 136
drwxr-xr-x  fff---  2 HAIMO    OMVSGRP     8192 May 29  2002 IBM
drwxr-xr-x  fff---  3 HAIMO    OMVSGRP     8192 Jun 10  2002 cat1
drwxrwxr-x  fff---  3 HAIMO    OMVSGRP     8192 May 29  2002 cat8
drwxr-xr-x  fff---  3 HAIMO    OMVSGRP     8192 May 29  2002 man1
-rw-rw-r--  fff---  2 HAIMO    OMVSGRP    33887 May 29  2002 whatis
```

Figure 6-76 Listing auditing options for specified files

The `ls` command with the `-W` option is used to display the auditing that is in effect for a file. In this example, since we have done the `ls` for a directory, all the files in the directory are displayed.

The auditing options are displayed to the right of the file permissions. The left three characters are for owner-set auditing, and the right three characters are for auditor-set auditing.

There are four possible characters for each of these columns:

- No auditing (`-`)
- Successful accesses (`s`)
- Access failures (`f`)
- All accesses (`a`)

6.77 Auditing reports

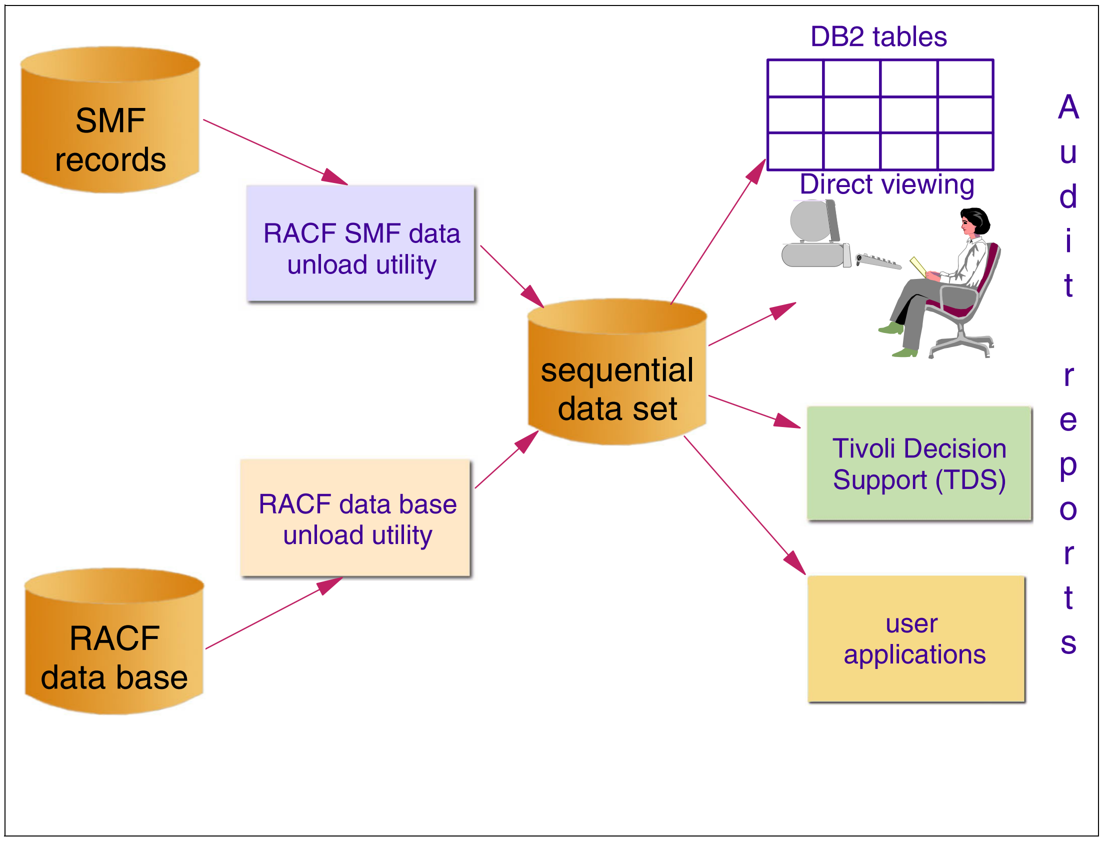


Figure 6-77 Creating auditing reports

In a z/OS UNIX environment which uses RACF as its access control program, the security auditor has two main tasks to perform:

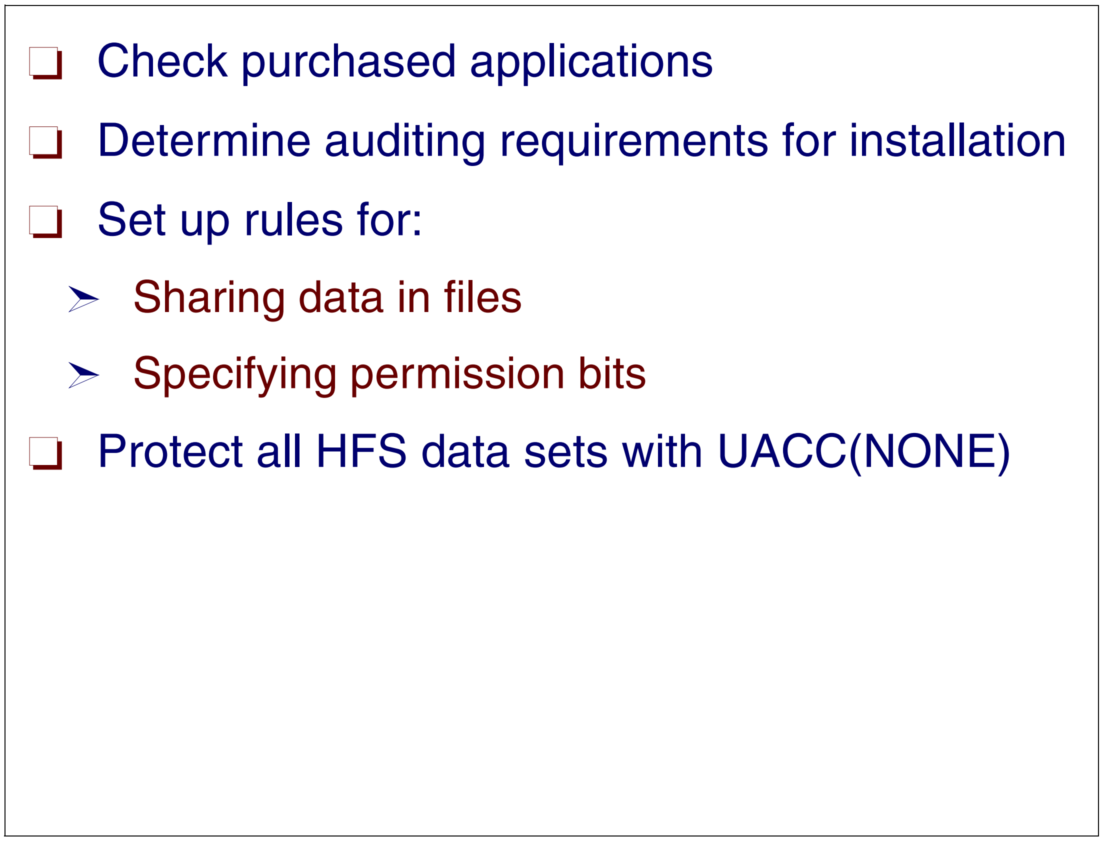
- Verify that the RACF profiles have the proper contents (OMVS segments in the user and group profiles, and logging options in particular).
- Use the security logs to follow up on detected violations, and to detect abnormal behavior by authorized users.

The audit information can be quite extensive and is found in the RACF database, the RACF security log, and in SMF records. RACF provides two utilities for unloading security data and placing the output in a sequential data set:

- The RACF database unload utility, IRRDBU00 can be used to unload a user's OMVS segment.
- The RACF SMF data unload utility, IRRADU00 can be used to unload the relevant audit records.

The output from these utilities can be used in several ways: viewed directly, used as input for installation-written programs, and manipulated with sort/merge utilities. It can also be uploaded to a database manager (for example, DB2®) to process complex inquiries and create installation-tailored reports. SYS1.SAMPLIB contains examples of how to create DB2 tables, and load RACF data produced from the RACF unload utilities, into the DB2 tables. It also has database query examples.

6.78 Maintain z/OS UNIX-level security



- Check purchased applications
- Determine auditing requirements for installation
- Set up rules for:
  - Sharing data in files
  - Specifying permission bits
- Protect all HFS data sets with UACC(NONE)

Figure 6-78 Checking products for z/OS UNIX-level security

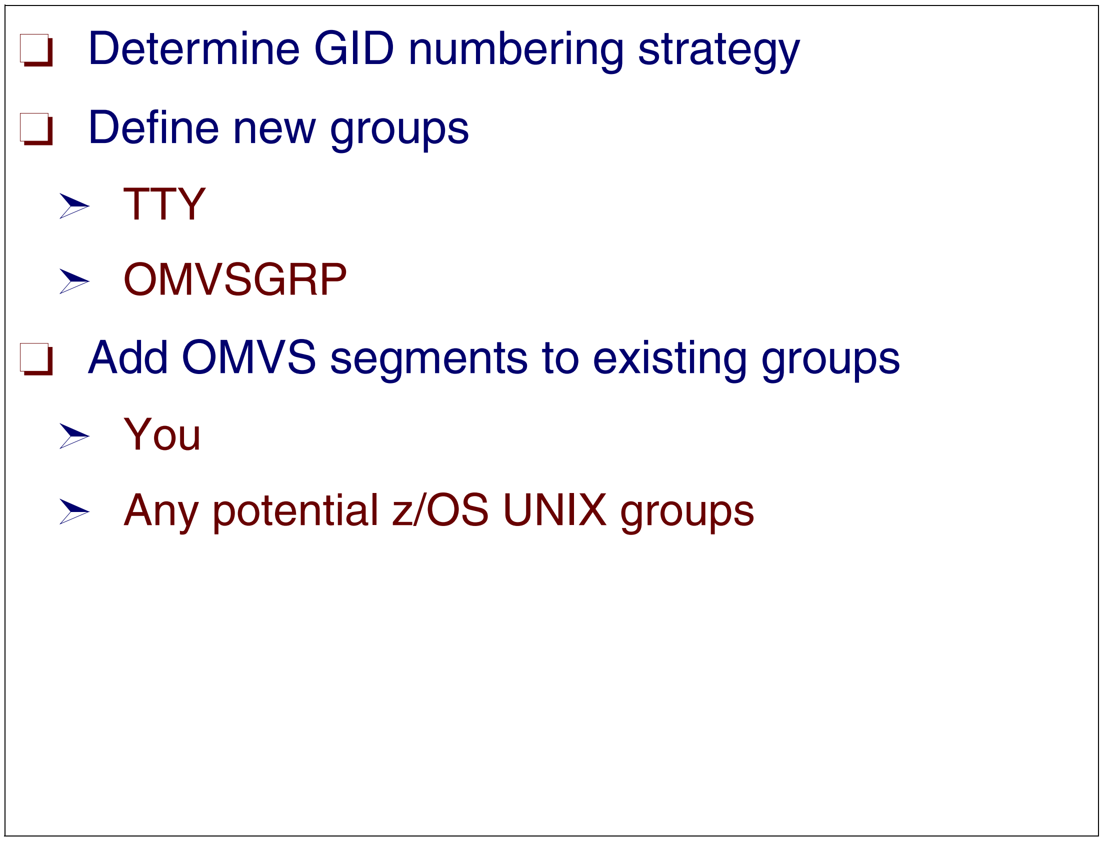
To maintain the security available from the system, take these steps:

- Check any purchased program to make sure that it will not lower the system security level. If you cannot be sure of a program, do not put it on the system.
- Determine the auditing requirements for the installation. Control auditing data with the RACF **`SETROPTS`** command and the **`chaudit`** shell command.
- Set up rules for:
  - Sharing data in files
  - Specifying permission bits when creating files or using the **`chmod`** shell command or **`chmod()`** function
- Protect all HFS data sets with a RACF profile that specifies `UACC(NONE)`. Only administrators with responsibility for creating, restoring, or dumping HFS data sets should be permitted to this profile.

To maintain security, require z/OS UNIX users to set the permission bits for their own files to deny access to any user but the file owner.

Each user is responsible for protecting their own data. However, data that other users need to access must have the appropriate permission bits set for group or other.

6.79 Setting up z/OS UNIX (1)



- Determine GID numbering strategy
- Define new groups
  - TTY
  - OMVSGRP
- Add OMVS segments to existing groups
  - You
  - Any potential z/OS UNIX groups

Figure 6-79 Tasks needed to set up z/OS UNIX

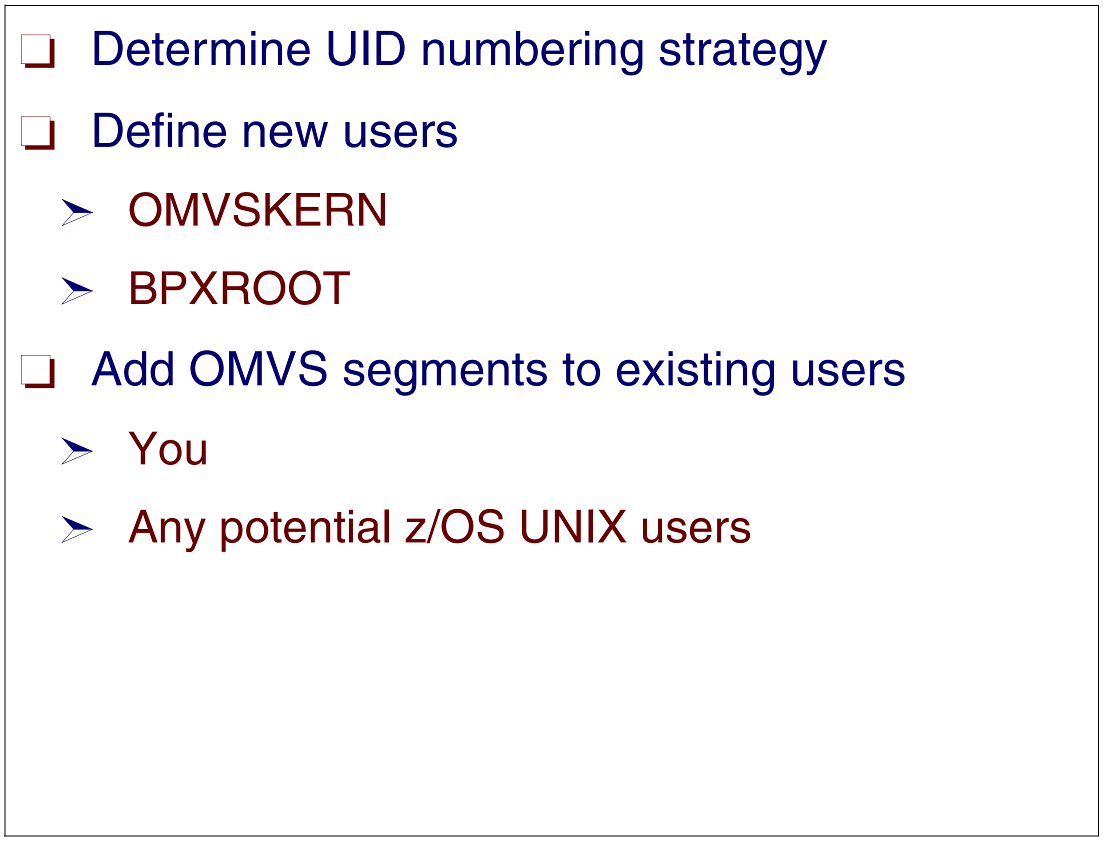
There is no predefined numbering scheme for z/OS UNIX GIDs. Therefore, you must decide what is suitable for your system when allocating GIDs.

Once a scheme has been chosen, you need to set up some groups. It is recommended that you set up at least two new groups: one for z/OS UNIX and another for TTY. The recommended group name for z/OS UNIX is OMVSGRP, but this can be changed to something else if it is not suitable on your system. The other group, TTY is recommended to be called that name—otherwise changes will be required in BPXPRMxx to support the different name.

You will need to add OMVS segments to some existing groups. For example, you (as the person doing the OMVS configuration work) will need a group OMVS segment (GID). Then you need to consider who else will need to use z/OS UNIX and ensure that they, too, have OMVS segments.

Some software detects that z/OS UNIX is active, and if so will want to use it. Two examples are TCP/IP and RMFGAT.

6.80 Setting up z/OS UNIX (2)



- ❑ Determine UID numbering strategy
- ❑ Define new users
  - ➢ OMVSKERN
  - ➢ BPXROOT
- ❑ Add OMVS segments to existing users
  - ➢ You
  - ➢ Any potential z/OS UNIX users

Figure 6-80 Tasks needed to set up z/OS UNIX

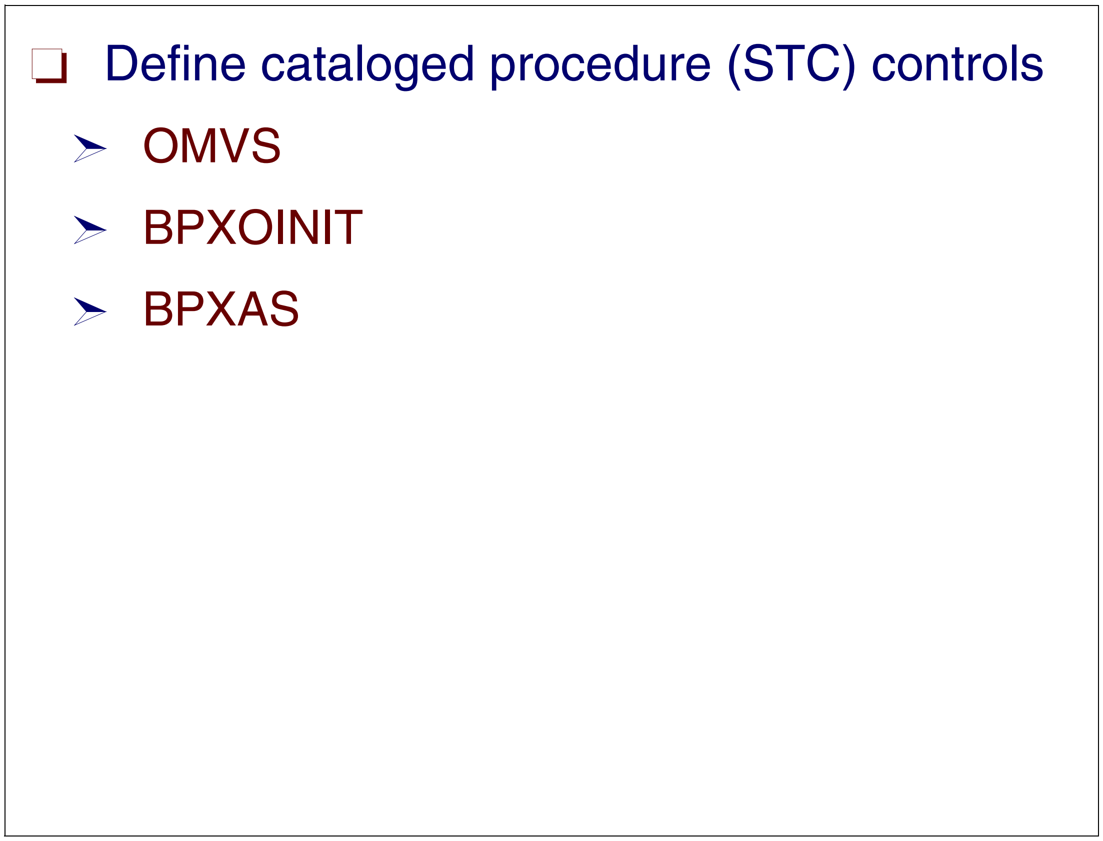
There is no predefined numbering scheme for z/OS UNIX UIDs other than UID=0 means Superuser. Therefore, you must decide what is suitable for your system when allocating UIDs.

Once a scheme has been chosen, you need to set up some user IDs. It is recommended that you set up at least two new user IDs: OMVSKERN and BPXROOT. The OMVSKERN user ID is used for the OMVS and BPXOINIT address spaces. The BPXROOT user ID is used when a daemon sets UID=0, but the user ID is not known. These recommended user ID names can be changed to something else if they are not suitable on your system.

You will need to add OMVS segments to some existing user IDs. For example, you (as the person doing the OMVS configuration work) will need a user ID OMVS segment (UID/HOME/PROGRAM). Then you need to consider who else will need to use z/OS UNIX and ensure that they, too, have OMVS segments.

Some software detects that z/OS UNIX is active, and if so will want to use it. Two examples are TCP/IP and RMFGAT.

6.81 Setting up z/OS UNIX (3)



- Define cataloged procedure (STC) controls
  - OMVS
  - BPXOINIT
  - BPXAS

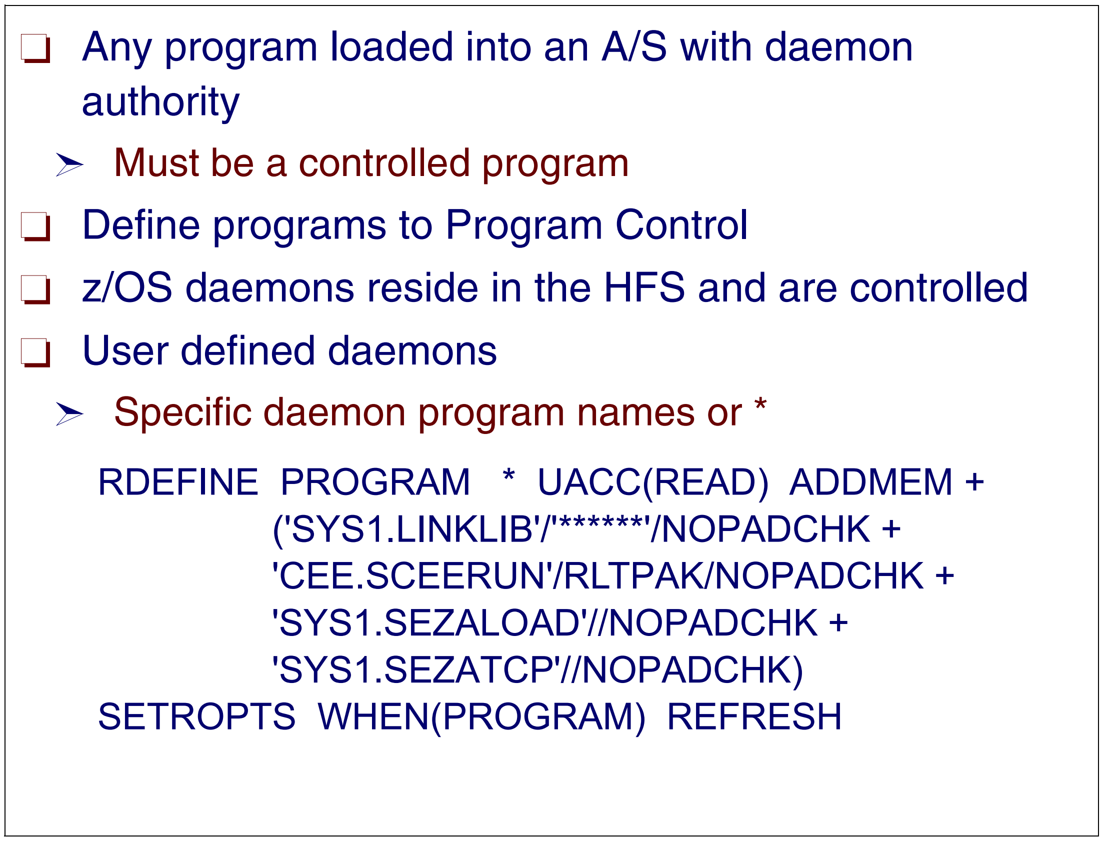
Figure 6-81 Tasks needed to set up z/OS UNIX

To activate z/OS UNIX, z/OS UNIX itself needs to have some identification (user ID/group) assigned to it. This is done either by the RACF started procedures table (ICHRIN03) or the STARTED class profiles.

Both the OMVS and BPXOINIT cataloged (or started) procedures (or started tasks - STCs) need to be defined. Only the OMVS procedure should be defined with the `TRUSTED` attribute.

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. Then it places the child process in a new address space. The forked address space is provided by workload manager (WLM), which uses the BPXAS PROC to create the address spaces.

6.82 Setting up z/OS UNIX (4)



- ❑ Any program loaded into an A/S with daemon authority
  - ➢ Must be a controlled program
- ❑ Define programs to Program Control
- ❑ z/OS daemons reside in the HFS and are controlled
- ❑ User defined daemons
  - ➢ Specific daemon program names or *

```
RDEFINE  PROGRAM   *  UACC(READ)  ADDMEM +
               ('SYS1.LINKLIB'/'******'/NOPADCHK +
               'CEE.SCEERUN'/RLTPAK/NOPADCHK +
               'SYS1.SEZALOAD'//NOPADCHK +
               'SYS1.SEZATCP'//NOPADCHK)
SETROPTS  WHEN(PROGRAM)  REFRESH
```

Figure 6-82 Tasks needed to set up z/OS UNIX

All programs loaded into an address space that requires daemon authority need to be marked as controlled (for example, user programs that are loaded and any run-time library modules that are loaded). All modules loaded from LPA are considered to be controlled. RTLS libraries must be defined to RACF for program-controlled support.

The BPX.DAEMON requires z/OS UNIX load libraries to be program-controlled. The data sets listed in Figure 6-82 are the main ones required to be program-controlled when setting up z/OS UNIX. These data sets are:

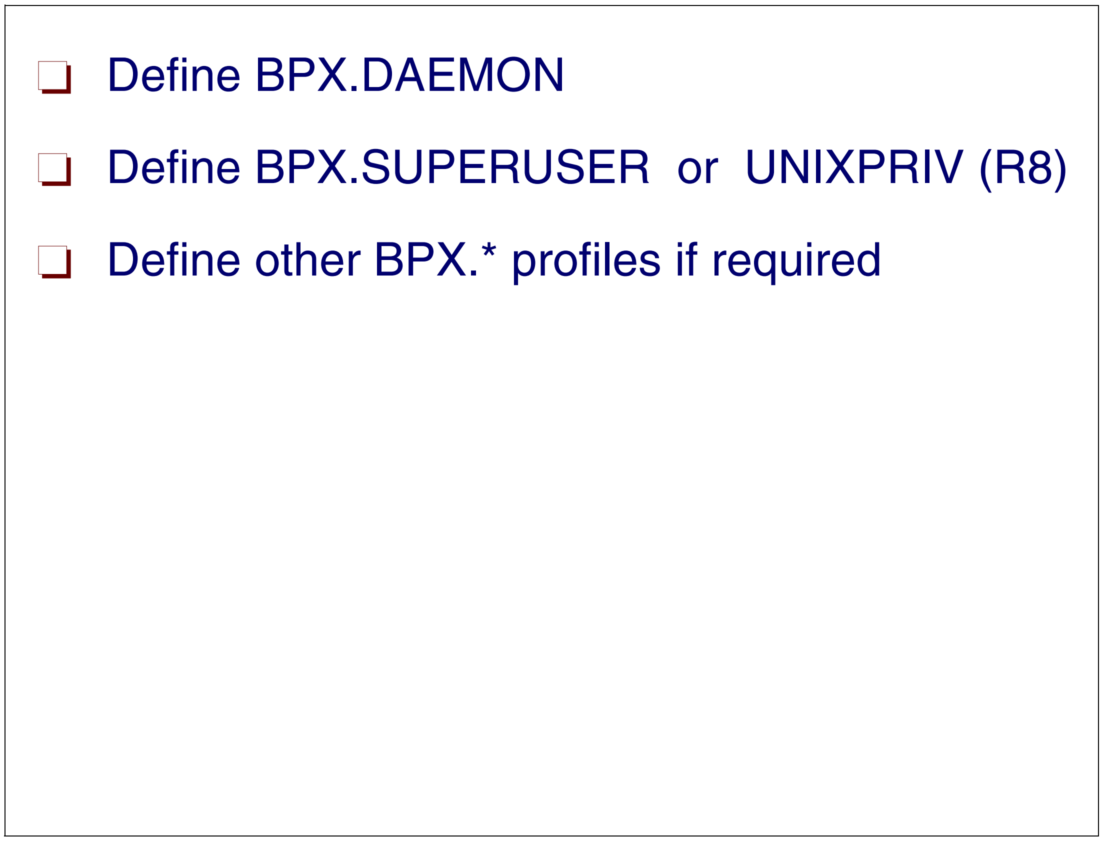
- LINKLIB - z/OS (including z/OS UNIX)
- SCEERUN - Language Environment
- SEZALOAD - TCP/IP
- SEZATCP - TCP/IP

Daemons shipped by z/OS reside in the HFS and are marked program-controlled, so you do not need to define them. 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 program-controlled:

```
RDEFINE PROGRAM SERVER1 ADDMEM('SYS1.LINKLIB'/'******'/NOPADCHK) + UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

Note: '******' (six asterisks surrounded by single quotes) specifies the current SYSRES volume.

6.83 Setting up z/OS UNIX (5)



- Define BPX.DAEMON
- Define BPX.SUPERUSER or UNIXPRIV (R8)
- Define other BPX.* profiles if required

Figure 6-83 Tasks needed to set up z/OS UNIX

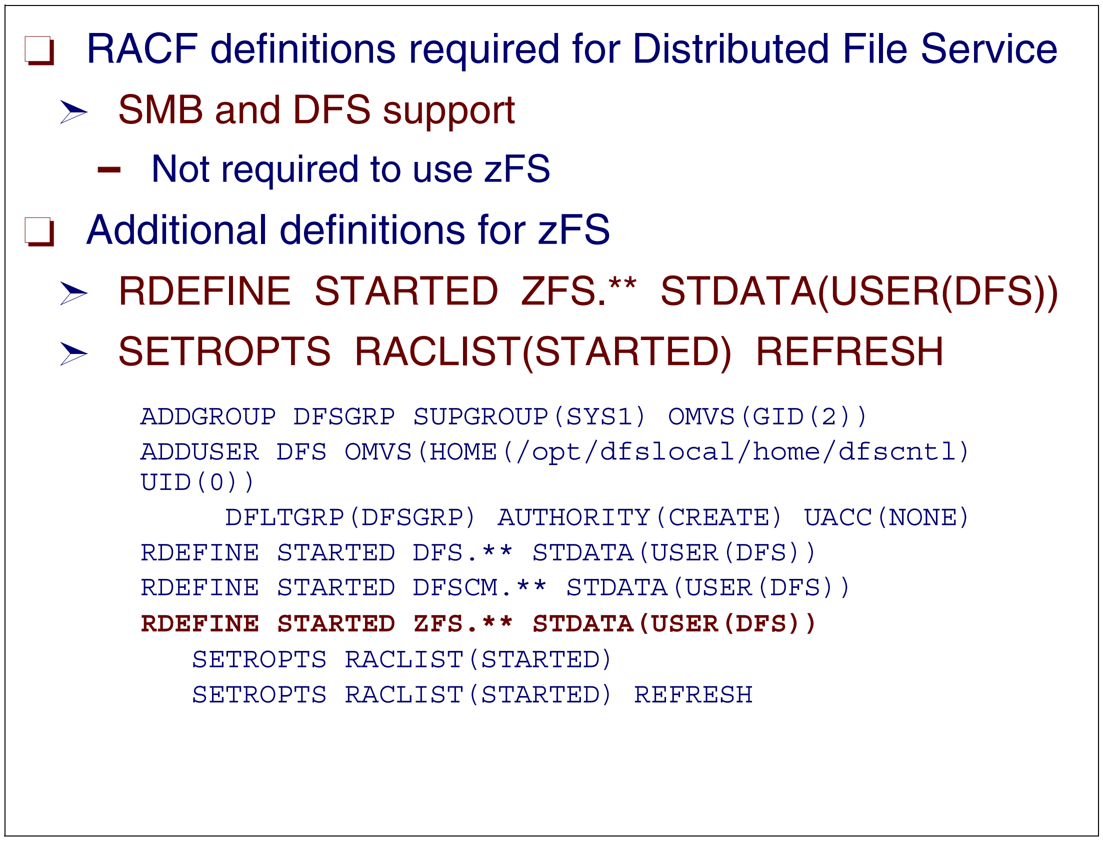
Of the BPX.* FACILITY class profiles, it is strongly recommended that at least BPX.DAEMON and BPX.SUPERUSER be implemented. BPX.DAEMON restricts access to the following services:

- seteuid (BPX1SEU service)
- setuid (BPX1SUI service)
- setreuid (BPX1SRU service)
- spawn (BPX1SPN service with a change in user ID requested.
- pthread_security_np()
- auth_check_resource_np()
- _login()
- _password()

BPX.SUPERUSER allows users with access to the BPX.SUPERUSER FACILITY class profile to switch to superuser authority (effective UID of 0).

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. These privileges are automatically granted to all users with z/OS UNIX superuser authority. 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 allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

6.84 RACF definitions for zFS



- RACF definitions required for Distributed File Service
  - SMB and DFS support
    - Not required to use zFS
- Additional definitions for zFS
  - RDEFINE STARTED ZFS.** STDATA(USER(DFS))
  - SETROPTS RACLIST(STARTED) REFRESH

```
ADDGROUP DFSGRP SUPGROUP(SYS1) OMVS(GID(2))
ADDUSER DFS OMVS(HOME(/opt/dfslocal/home/dfscntl)
UID(0))
     DFLTGRP(DFSGRP) AUTHORITY(CREATE) UACC(NONE)
RDEFINE STARTED DFS.** STDATA(USER(DFS))
RDEFINE STARTED DFSCM.** STDATA(USER(DFS))
RDEFINE STARTED ZFS.** STDATA(USER(DFS))
   SETROPTS RACLIST(STARTED)
   SETROPTS RACLIST(STARTED) REFRESH
```

Figure 6-84 RACF definitions for zFS

To define DFS to RACF you must create the following definitions with these exact names. Even if you use none of the functions of DFS, you can use the following DFS definitions for zFS:

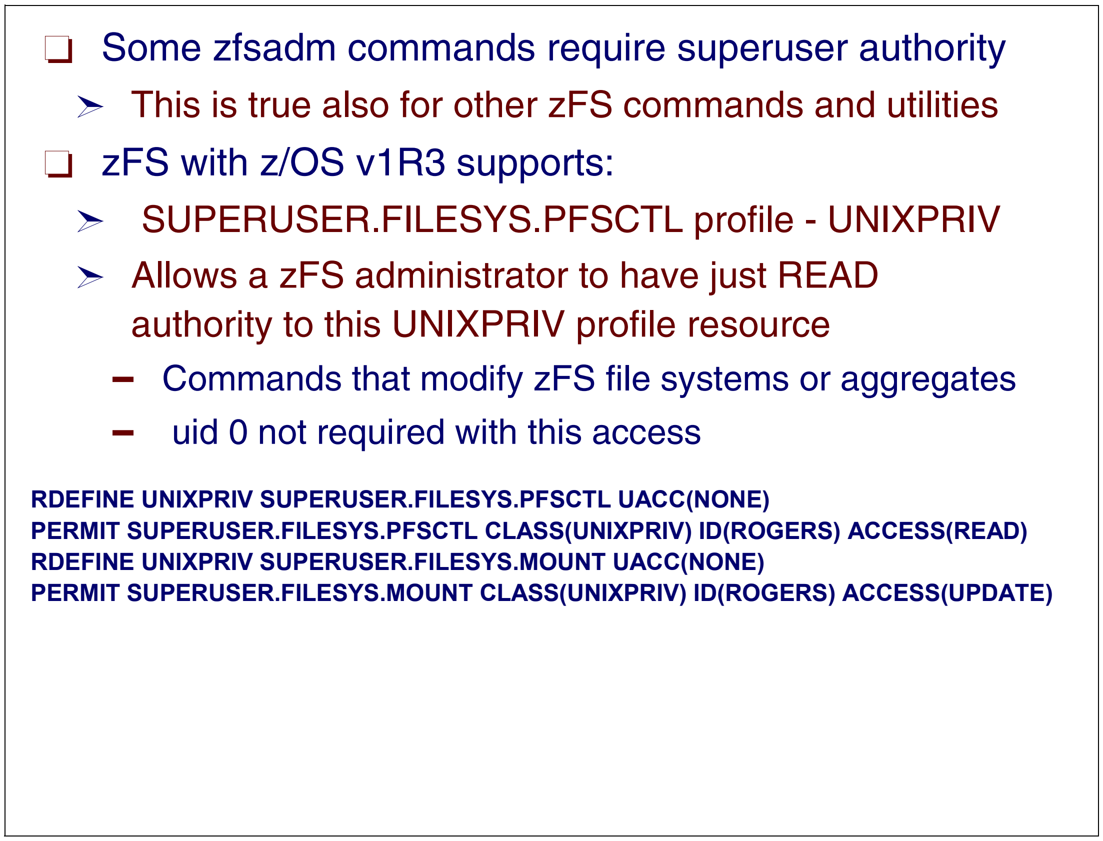
- Define DFSGRP as a group.
- Define DFS as a user.
- Define ZFS as a started task.

For RACF, use the following commands:

```
ADDGROUP DFSGRP SUPGROUP(SYS1) OMVS(GID(2))
ADDUSER DFS OMVS(HOME(/opt/dfslocal/home/dfscntl) UID(0))
     DFLTGRP(DFSGRP) AUTHORITY(CREATE) UACC(NONE)
RDEFINE STARTED ZFS.** STDATA(USER(DFS) GROUP(DFSGRP))
SETROPTS RACLIST(STARTED) REFRESH
```

Note: A user ID other than DFS can be used to run the ZFS started task if it is defined with the same RACF characteristics as shown for the DFS user ID.

6.85 UNIXPRIV class with z/OS V1R3 and zFS



- Some zfsadm commands require superuser authority
  - This is true also for other zFS commands and utilities
- zFS with z/OS v1R3 supports:
  - SUPERUSER.FILESYS.PFSCTL profile - UNIXPRIV
  - Allows a zFS administrator to have just READ authority to this UNIXPRIV profile resource
    - Commands that modify zFS file systems or aggregates
    - uid 0 not required with this access

RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(ROGERS) ACCESS(UPDATE)

Figure 6-85 UNIXPRIV class changes for zFS

zFS with z/OS V1R3 supports the SUPERUSER.FILESYS.PFSCTL profile of the UNIXPRIV class. This makes it possible for a zFS administrator to have just READ authority to this UNIXPRIV profile resource, SUPERUSER.FILESYS.PFSCTL, rather than requiring a UID of 0 for **zfsadm** commands that modify zFS file systems or aggregates. The same is true for the other zFS commands and utilities, so zFS administrators do not need a UID of 0.

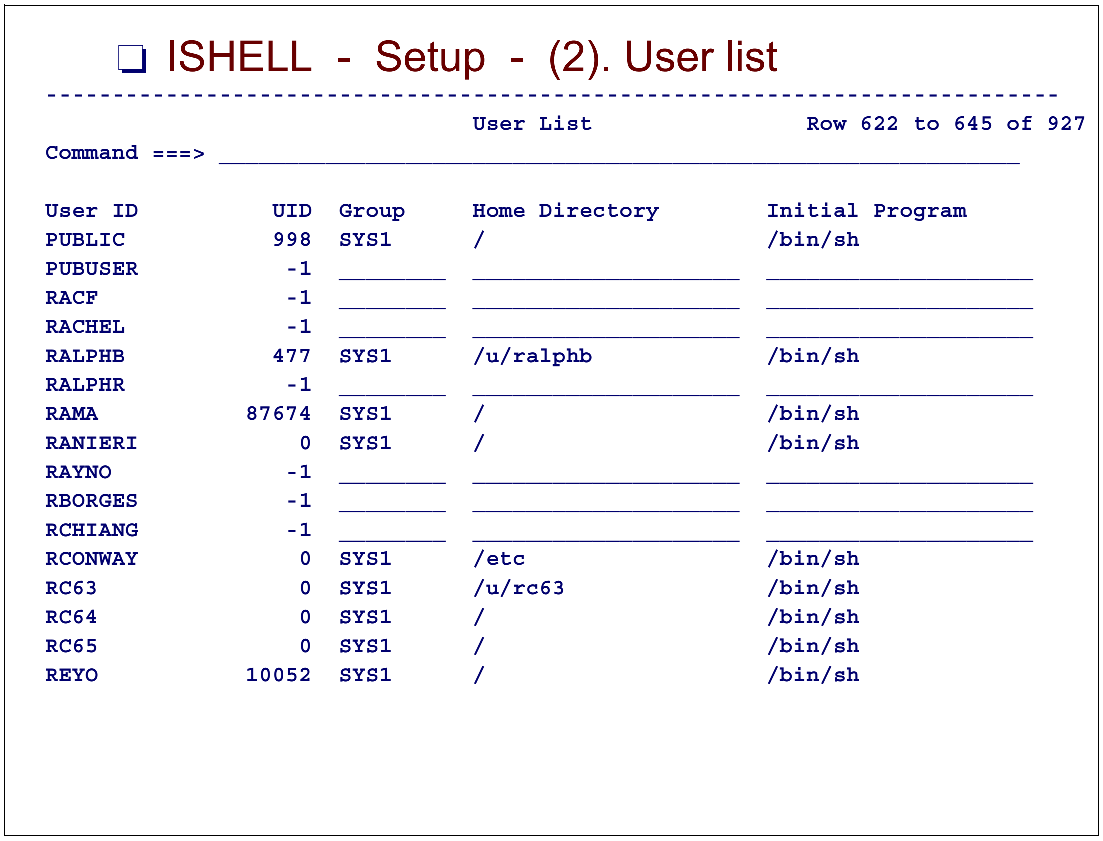
To allow the zFS administrator to mount and unmount file systems, permit update access to another profile, SUPERUSER.FILESYS.MOUNT, in class UNIXPRIV.

Note: UPDATE access is needed if the user needs to **mount**, **chmount**, or **unmount** file systems with the `setuid` option; otherwise, READ access is sufficient.

UNIXPRIV authorization is invoked by creating the needed resources in the UNIXPRIV class and then giving users READ authority to it, as follows:

```
SETROPTS CLASSACT(UNIXPRIV)
SETROPTS RACLIST(UNIXPRIV)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(ROGERS) ACCESS(UPDATE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

6.86 List current users using the ISHELL


❑ ISHELL - Setup - (2). User list

```
---------------------------------------------------------------------------
                              User List                Row 622 to 645 of 927
Command ===> ____________________________________________________________

User ID          UID  Group     Home Directory       Initial Program
PUBLIC           998  SYS1      /                    /bin/sh
PUBUSER           -1  ________  ____________________ ____________________
RACF              -1  ________  ____________________ ____________________
RACHEL            -1  ________  ____________________ ____________________
RALPHB           477  SYS1      /u/ralphb            /bin/sh
RALPHR            -1  ________  ____________________ ____________________
RAMA           87674  SYS1      /                    /bin/sh
RANIERI            0  SYS1      /                    /bin/sh
RAYNO             -1  ________  ____________________ ____________________
RBORGES           -1  ________  ____________________ ____________________
RCHIANG           -1  ________  ____________________ ____________________
RCONWAY            0  SYS1      /etc                 /bin/sh
RC63               0  SYS1      /u/rc63              /bin/sh
RC64               0  SYS1      /                    /bin/sh
RC65               0  SYS1      /                    /bin/sh
REYO           10052  SYS1      /                    /bin/sh
```

Figure 6-86 List the currently defined z/OS UNIX users

A system programmer can use the ISPF shell to perform tasks such as listing the current users defined in the OMVS segment. This requires superuser authority.

7

Managing file systems

This chapter describes the hierarchical file system and how to customize it to your requirements. In addition to providing an introduction to HFS concepts, it provides details on how to:

- Manage and create a hierarchical file system
- Use commands to display useful information
- Perform space management for a hierarchical file system

7.1 Hierarchical file system

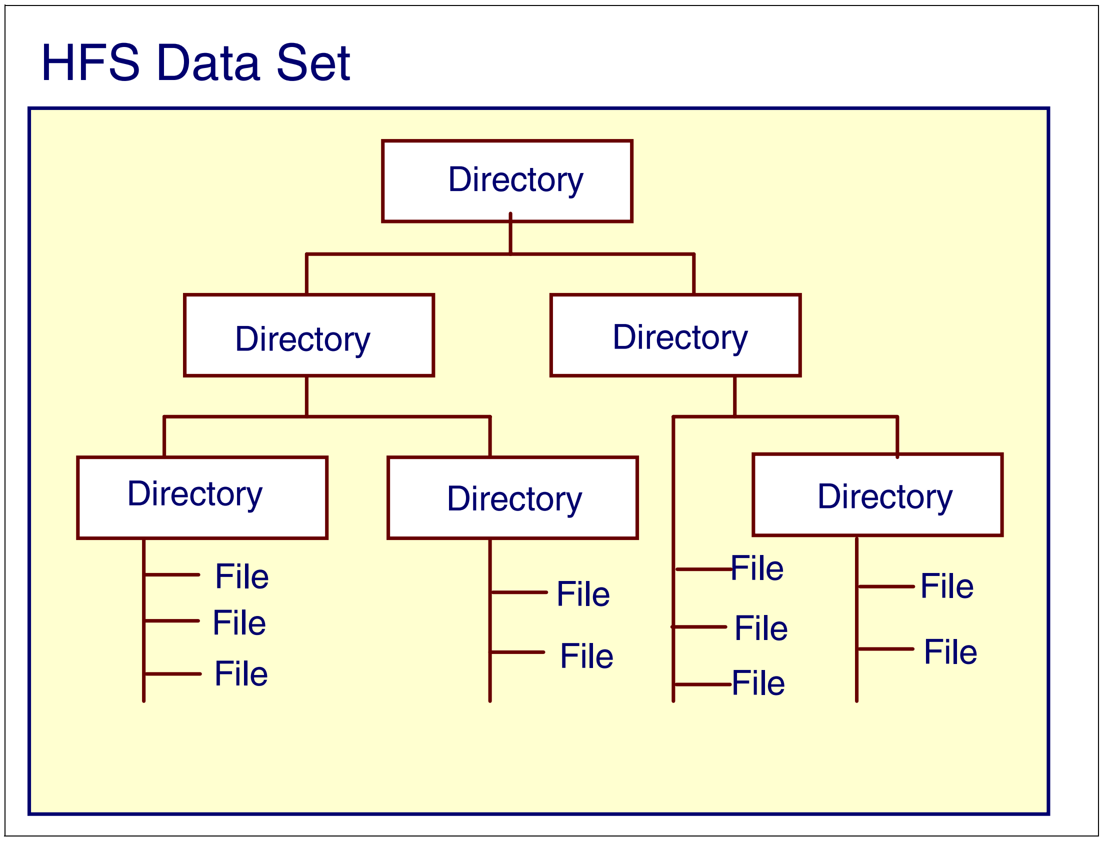


Figure 7-1 Hierarchical file system structure

The z/OS UNIX file system is hierarchical and byte-oriented. Finding a file in the file system is done by searching a directory or a series of directories. There is no concept of a z/OS catalog that points directly to a file.

A path name identifies a file and consists of directory names and a file name. A fully qualified file name, which consists of the name of each directory in the path to a file plus the file name itself, can be up to 1023 bytes long, for example: /u/joe/projects/p1/prog1.

The hierarchical file system allows for file names in mixed case.

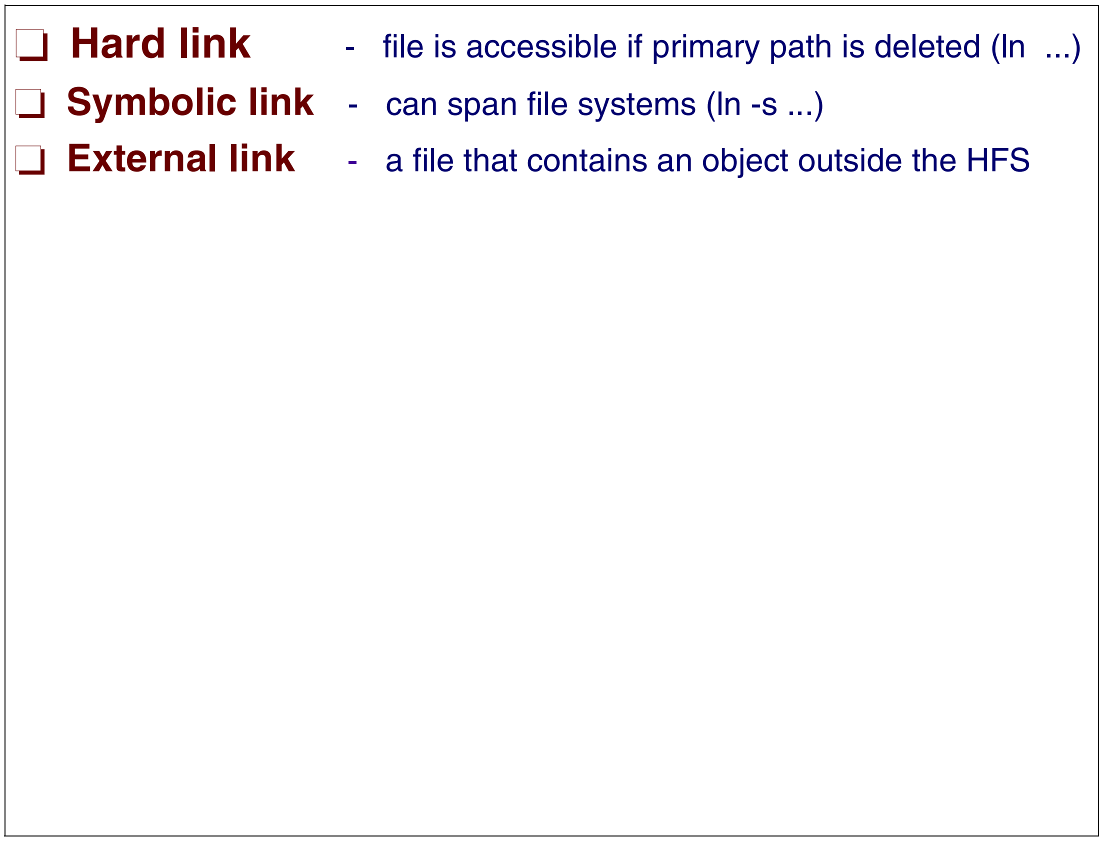
The Hierarchical File System (HFS) data set, which contains the hierarchical file system, is a new z/OS data set type.

The files in the hierarchical file system are sequential files, and they are accessed as byte streams. A record concept does not exist with these files, other than the structure defined by an application.

The files can be text files or binary files. In a text file, each line is separated by a newline delimiter. A binary file consists of sequences of binary words, and the application reading or writing to the file is responsible for the format of the data.

A file name can be up to 255 characters long. HFS data sets and z/OS data sets can reside on the same SMS-managed DASD volume.

7.2 File linking



- **Hard link** - file is accessible if primary path is deleted (ln ...)
- **Symbolic link** - can span file systems (ln -s ...)
- **External link** - a file that contains an object outside the HFS

Figure 7-2 File linking

A link is a new pathname, or directory entry, for an existing file. The new directory entry can be in the same directory that holds the file, or in a different directory. You can access the file under the old pathname or the new one. After you have a link to a file, any changes you make to the file are evident when it is accessed under any other name. Links are useful when:

- A file is moved and you want users to be able to access the file under the old name.
- You want to create a link with a short pathname (an alias) for a file that has a long pathname.

You can use the `ln` command to create a hard link or a symbolic link. A file can have an unlimited number of links to it.

Note: Hard links work only on the same HFS. Use symbolic links to span HFSes.

7.3 Hard links

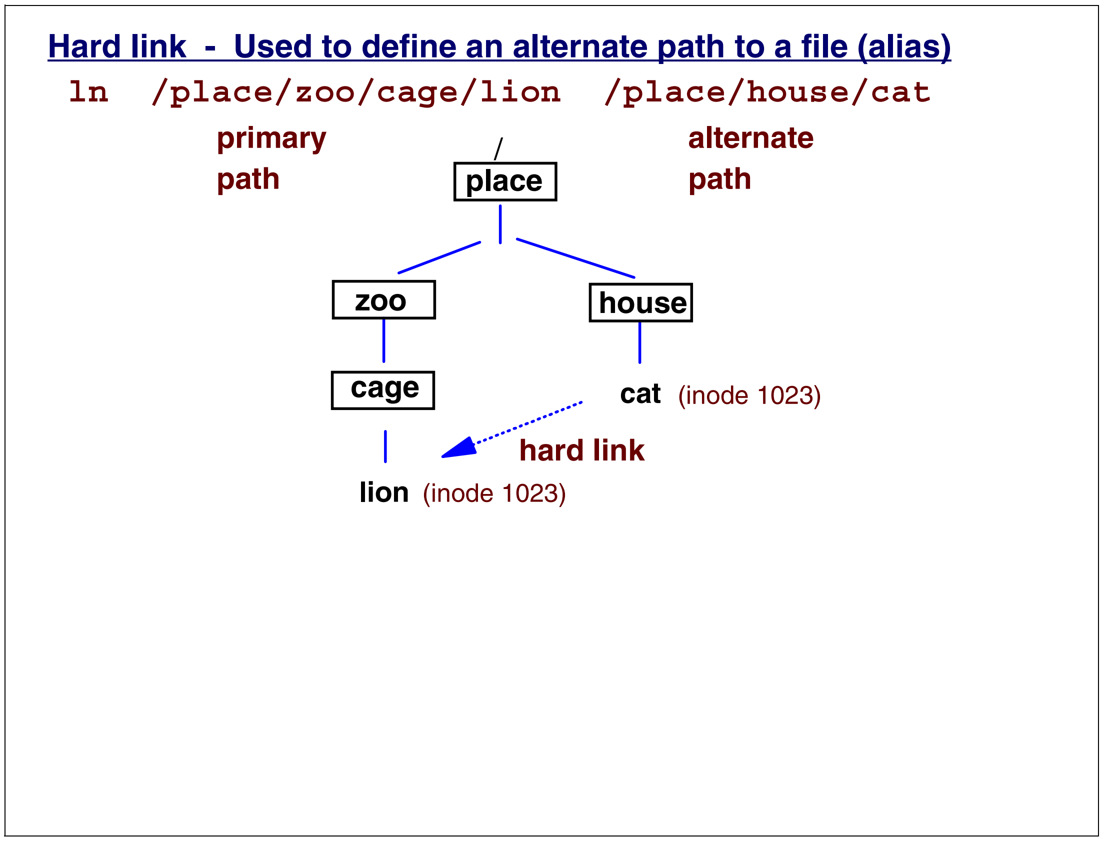

Hard link - Used to define an alternate path to a file (alias)
ln /place/zoo/cage/lion /place/house/cat
primary path
alternate path
/
place
zoo
house
cage
cat (inode 1023)
hard link
lion (inode 1023)

Figure 7-3 Hard link example

A hard link is a new name for an existing file. You cannot create a hard link to a directory, and you cannot create a hard link to a file on a different mounted file system.

All the hard link names for a file are as important as its original name. They are all real names for the one original file. To create a hard link to a file, use this command format:

```
ln old new
```

When you create a hard link to a file, the new filename shares the inode number of the original physical file. Because an inode number represents a physical file in a specific file system, you cannot make hard links to other mounted file systems.

7.4 Symbolic links

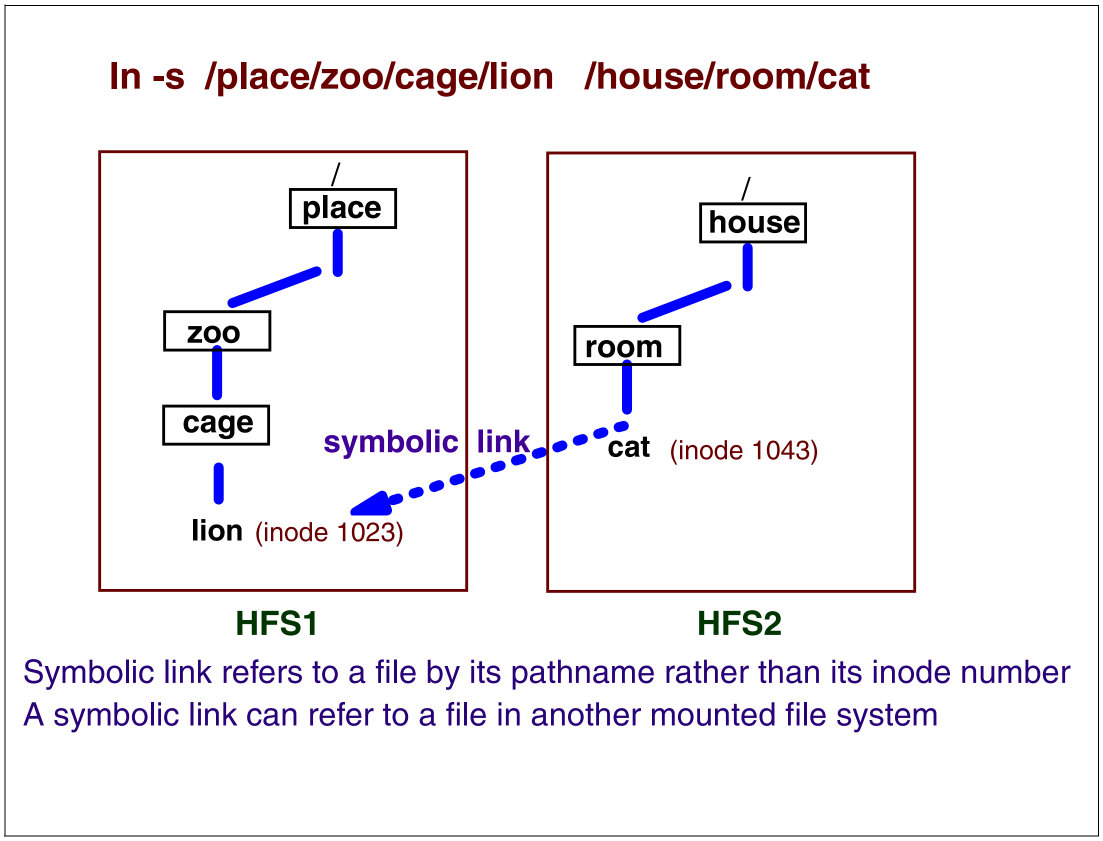

ln -s /place/zoo/cage/lion /house/room/cat
/
place
zoo
cage
lion (inode 1023)
symbolic link
/
house
room
cat (inode 1043)
HFS1
HFS2
Symbolic link refers to a file by its pathname rather than its inode number
A symbolic link can refer to a file in another mounted file system

Figure 7-4 Symbolic link examples

A symbolic link is a type of file system entry that contains the pathname of and acts as a pointer to another file or directory.

You can create a symbolic link to a file or a directory. Additionally, you can create a symbolic link across mounted file systems, which you cannot do with a hard link. A symbolic link is another file that contains the pathname for the original file—in essence, a reference to the file. A symbolic link can refer to a pathname for a file that does not exist. To create a symbolic link to a file, use this command format:

```
ln -s old new
```

where “new” is the name of the new file containing the reference to the file named “old.” In Figure 7-4, /house/room/cat is the name of the new file that contains the reference to /place/zoo/cage/lion.

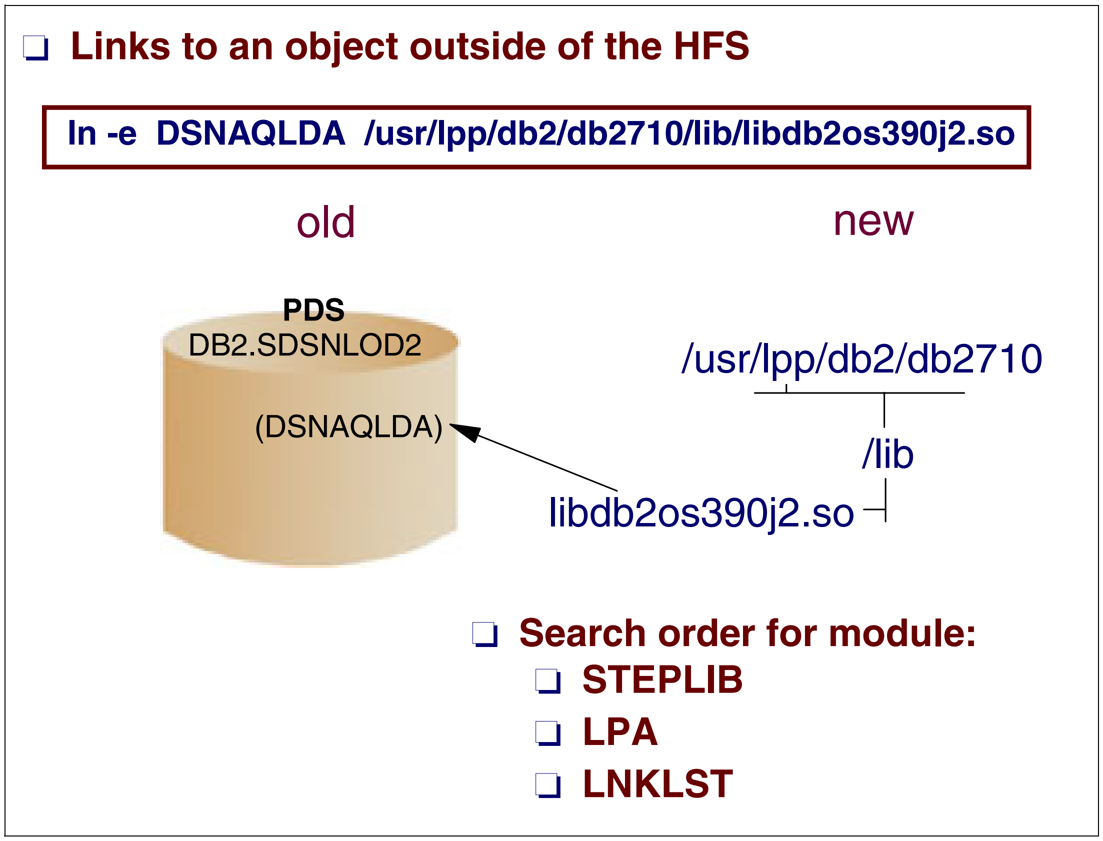
When you create a symbolic link, you create a new physical file with its own inode number, as shown in the figure. Because a symbolic link refers to a file by its pathname rather than by its inode number, a symbolic link can refer to files in other mounted file systems.

INODE

The *inode* is the internal structure that describes the individual files in the operating system; there is one inode for each file. An inode contains the node, type, owner, access times, number of links, and location of a file. A table of inodes is stored near the beginning of a file system. The 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 time stamps. 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.

The inode number specifies a particular inode file in the file system.

7.5 External links



❑ Links to an object outside of the HFS

ln -e DSNAQLDA /usr/lpp/db2/db2710/lib/libdb2os390j2.so

❑ Search order for module:
- ❑ STEPLIB
- ❑ LPA
- ❑ LNKLST

Figure 7-5 Defining external links

An external link is a special type of symbolic link. It is a file that contains the name of an object that is outside of the hierarchical file system.

DFSMS/NFS uses this link to access z/OS data sets.

An example of defining an external link is to make a link between an HFS file and an MVS program, as follows:

- Define an HFS file as:

```
/usr/lpp/internet/bin/wwwss.so
```

- Where the MVS program name IMWYWWS is an external link, the command to define the external link is:

```
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.

Using an external link, you associate that object with a pathname. For example, **`setlocale()`** searches for locale object files in the HFS, but if you want to keep your locale object files in a partitioned data set, you can create an external link in the HFS that points to the PDS. This will improve performance by shortening the search that **`setlocale()`** makes.

A file can be an external link to a sequential data set, a PDS, or a PDS member.

Consider the following examples:

```
ln -e SYS1.PRIVATE.PGMA   /u/ggi/plib/pgma
ln -e PGMPDS              /u/ggi/plib/pgm
```

The second example assumes that the PGMPDS member will be found in a PDS library, using the traditional z/OS search order. Because of performance considerations, put the library into the LPA or LNKLST.

7.6 Root file system

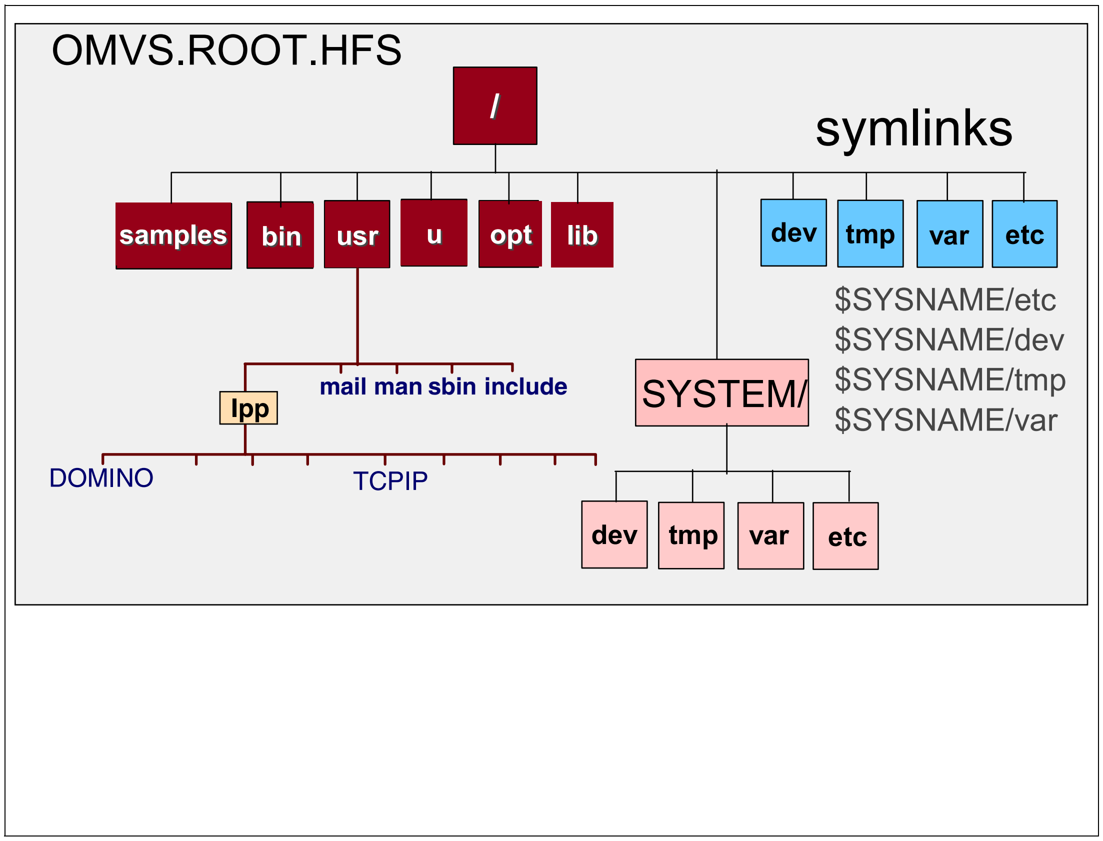


Figure 7-6 The root file system

The root file system is the starting point for the overall file system structure. It consists of the root (/) directory, system directories, and files. A system programmer defines the root file system. The system programmer must have an OMVS UID of 0 to allocate, mount, and customize the root directories. It usually contains system-related files and files that belong to a program product.

The z/OS UNIX root file system directories and their contents are as follows:

bin	Contains executable modules, mostly shell commands.
var	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 during execution of code. An example of this is caching data.
dev	Contains character-special files that are used when logging into the OMVS shell environment and also during c89 processing. Prior to OS/390 V2R7, these character-special files were created by running the BPXISMKD REXX exec or would be part of your ServerPac order. Beginning with OS/390 V2R7, /dev is shipped empty. The necessary files are created when the system is IPLed, and on a per-demand basis.
etc	Contains customization data. Keeping the /etc file system in an HFS data set 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. After you complete instructions for a ServerPac or CBPDO installation, you will have an /etc file system in its own HFS data set.

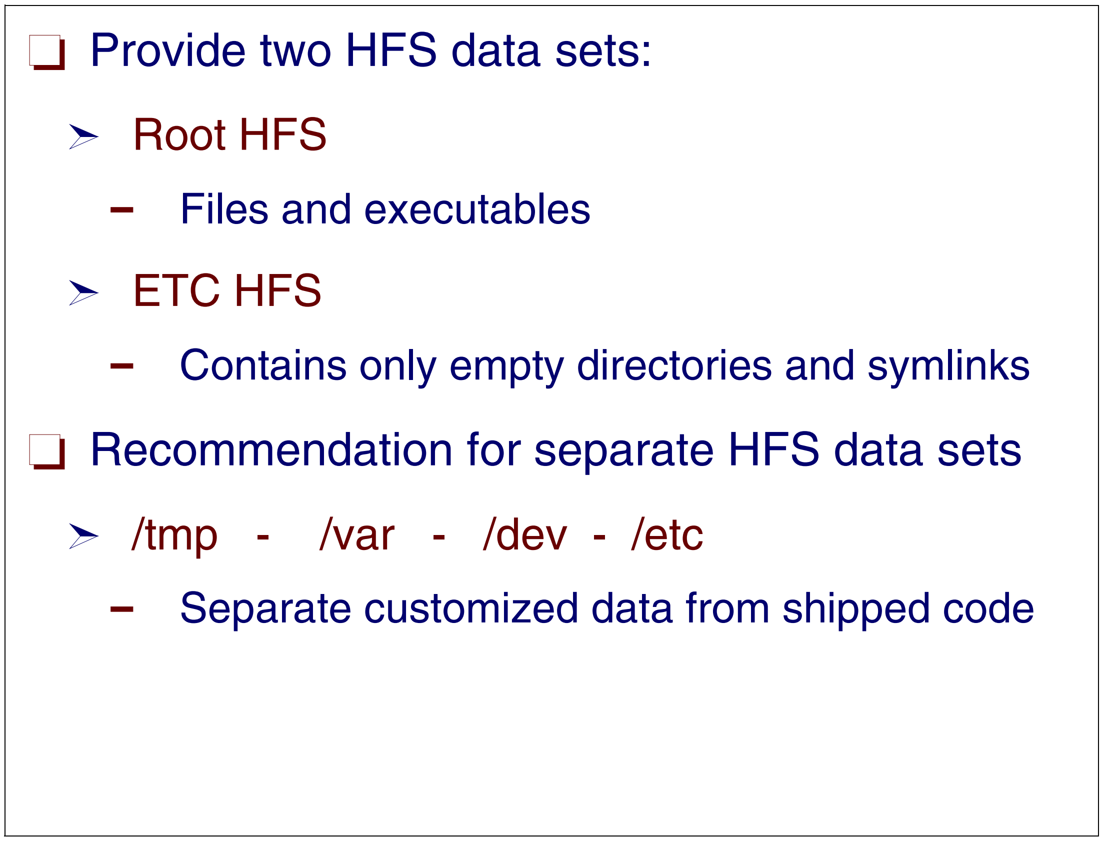
tmp	Contains temporary data that is used by products and applications. The directory /tmp is created empty, and temporary files are created dynamically by different elements and products. You have the option of mounting a temporary file system (TFS) on /tmp.
lib	This directory contains symbolic links to the TCP/IP directory for the X11 Window interface. In a z/OS UNIX system, it is used as a library containing C run-time libraries.
samples	Contains examples of files that can be customized and used in the /etc directory.
u	Contains user home directories
usr/lpp	Contains subdirectories for other applications like Domino®, WebSphere, TCP/IP, DCE, daemons, and so on.
SYSTEM	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.

Symbolic links (Symlinks)

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

Note: 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.

7.7 ServerPac and CBPDO



- Provide two HFS data sets:
  - Root HFS
    - Files and executables
  - ETC HFS
    - Contains only empty directories and symlinks
- Recommendation for separate HFS data sets
  - /tmp - /var - /dev - /etc
    - Separate customized data from shipped code

Figure 7-7 ServerPac and the root file system

For z/OS ServerPac customers, IBM delivers a single-root HFS. This HFS is unloaded when you perform the “Establishing UNIX Services” step in the ServerPac installation process. Not only does the single-root HFS make cloning of file systems easier, but it also dramatically reduces the number of jobs run by system programmers to establish z/OS UNIX.

IBM also delivers a separate HFS data set for /etc.

Performing ServerPac installation requires that you be a superuser with UID(0) or have access to the BPX.SUPERUSER resource in the FACILITY class.

For customers who use the Custom-Built Product Delivery Option (CBPDO) software delivery package, a sample job called BPXISHFS (found in SYS1.SAMPLIB) is provided. It allocates the root and /etc HFS data sets and then mounts them at a given mount point. This job allocates a file system that is mounted on the /etc directory so that z/OS-delivered code is part of a single HFS, while customized data can be kept separate. This allows for easier cloning of file systems.

/etc file system

The /etc file system contains customization data, such as the definition files for the automount facility. The install process creates an empty /etc directory into which customers must copy their existing /etc file system. This directory is similar to SYS1.PARMLIB, but differs in some aspects. For example, the /etc file system cannot be shared between systems, nor can the /etc file system be concatenated with other directories like SYS1.PARMLIB can. To keep your customization data separated from IBM's service updates and to make migration to another release easier, keep the /etc file system in an HFS data set separate from other file systems. When you complete all instructions for installing z/OS, whether through a ServerPac or CBPDO, you have an /etc file system in its own HFS data set.

To ensure that your customization files are not modified, IBM does not create any files in the /etc file system. IBM does, however, create directories when they are needed. IBM also doesn't ship maintenance into /etc via SMP/E.

Starting in OS/390 V2R9, place the contents of the /etc, /dev, /tmp, and /var directories for each system into separate HFS data sets if they have not already been set up that way. This task is required for both non-sysplex users and sysplex users.

7.8 Recommended customization

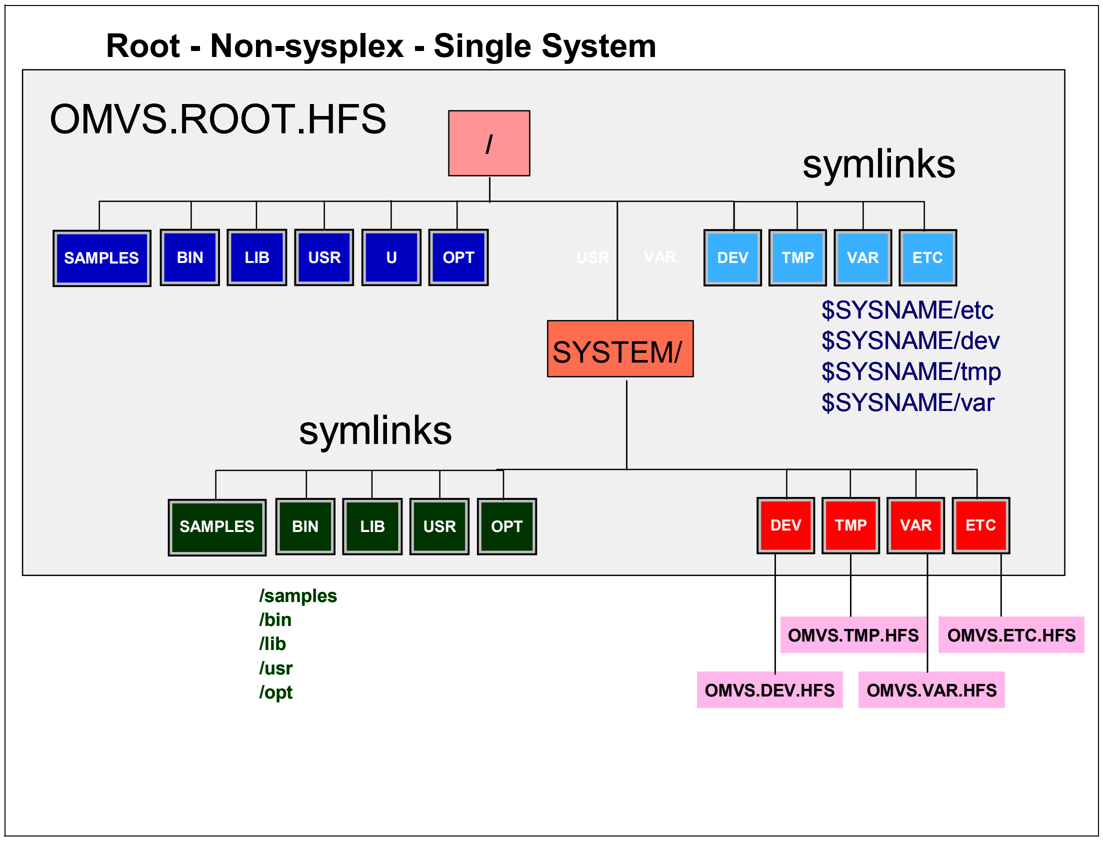


Figure 7-8 Recommended customization of the root file system

In Figure 7-8, 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. During installation, IBM defines directories in the /etc directory but does not install files there. Because the configuration and customization data in your existing /etc directory might not be correct for the new system, you might need to make changes to the files in your new /etc directory. IBM recommends that you make these changes before the first IPL of the new system. The presence of symbolic links is transparent to the user.

Starting in OS/390 V2R9, place the contents of the /etc, /dev, /tmp, and /var directories for each system into separate HFS data sets if they have not already been set up that way. This task is required for both non-sysplex users and sysplex users.

Some utilities that are provided by z/OS UNIX require the use of certain configuration files. Customers are responsible for providing these files if they expect to use these utilities at their installation. IBM provides default configuration files as samples in the /samples directory. Before the first use of any of these utilities, customers must copy these IBM-provided samples to the /etc directory (in most cases). Further customization of these files to include installation-dependent information can be added by the customer.

Note: 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.

7.9 File system structure

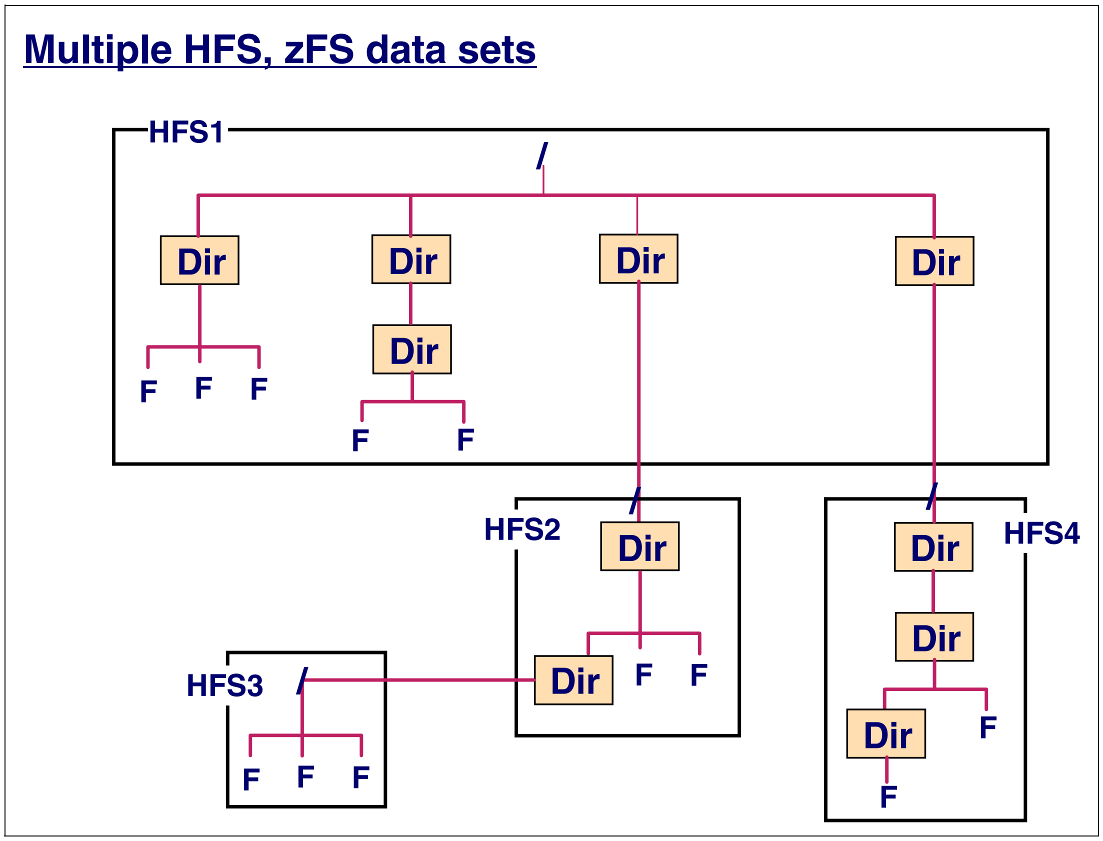


Figure 7-9 File system structure

The z/OS UNIX file system can consist of multiple file systems joined together to form a hierarchical file system. Connecting one file system to another is called *mounting*.

The root file system is the first file system that is mounted, and subsequent file systems can be mounted on a directory in a file system that is already mounted. The file systems will form a hierarchy of file systems. A file system must be mounted before anybody can access it.

An HFS data set is a mountable file system. A mountable file system can be logically mounted to a mount point, which is a directory in another file system, with a TSO/E **MOUNT** command. A mountable file system can be unmounted from a mount point with a TSO/E **UNMOUNT** command. ISPF users can also perform these tasks using the ISPF shell.

After the installation, create mountable file systems and mount them to your root file system, or elsewhere in the hierarchy.

Build a directory in the root file system. A directory can be used as a mount point for a mountable file system. The mount point should be an empty directory; otherwise, its contents will be hidden for the duration of any subsequent mounts.

Note: If you want to unmount HFS2, you first have to unmount HFS3 or the system will tell you that the resource is busy. However, you can unmount by using the immediate option or the force option without first unmounting HFS3. If users are working on those directories, they will get errors trying to list the current directories.

7.10 Temporary directory space

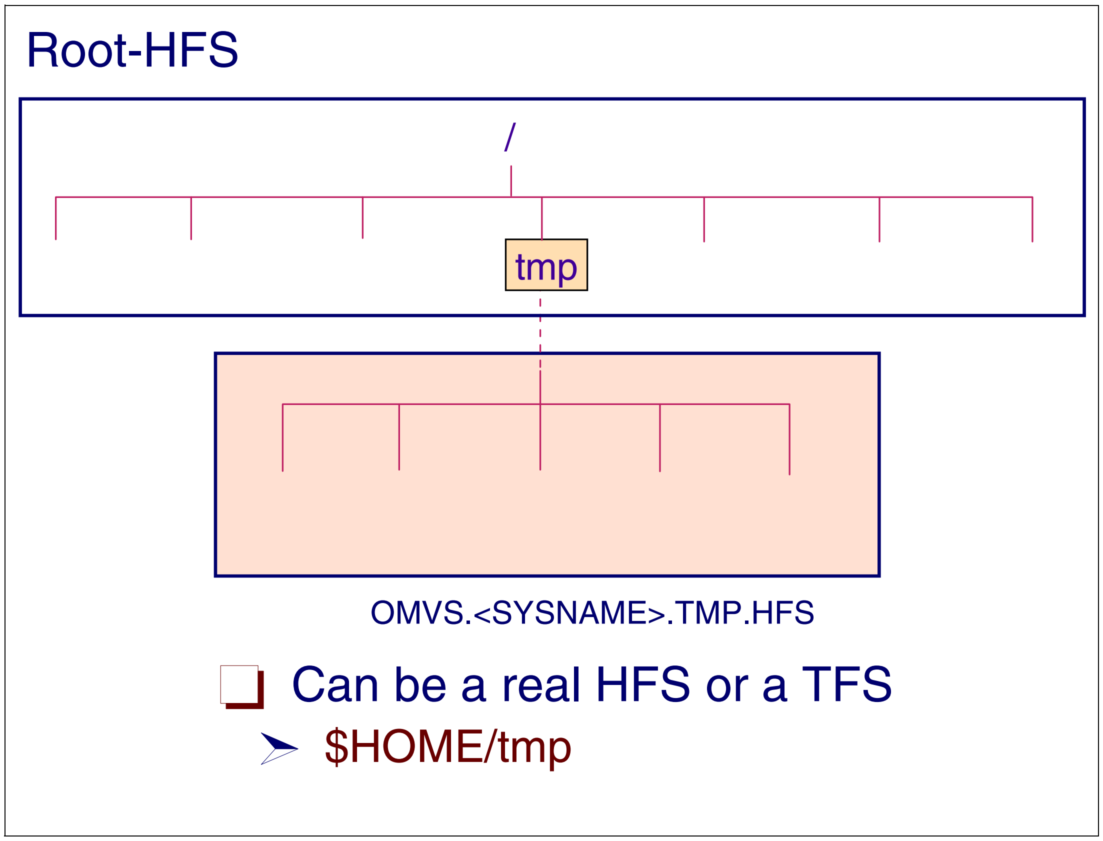


Figure 7-10 Temporary file system and temporary directory space

When compiling C programs, the compiler will use the /tmp directory for temporary files. When compiling large applications, the /tmp directory can use a lot of space. Use a separate file system for temporary data. In this way, a maximum of one volume will be used for temporary space.

Placing the /tmp directory in a separate file system can also improve performance in a system with a large number of interactive z/OS UNIX users, where the I/O activity can be very high on the /tmp directory. Do this with the following procedure:

- Allocate an HFS data set for the temporary data file system, for example:

  ```
  OMVS.<SYSNAME>.TMP.HFS
  ```

- Mount this data set on the /tmp directory in the root file system. When data is written to files in the /tmp directory, the data will be physically located in the OMVS.<SYSNAME>.TMP.HFS file system.

When you have a large number of interactive users, the /tmp directory can sustain large amounts of I/O activity. There are several approaches you can take:

- Mount a temporary file system (TFS) over /tmp in the HFS, so that you have a high-speed file system for temporary files. The temporary file system is an in-memory file system that is not written to DASD.
- Place the /tmp directory in its own mountable file system and put the file system on its own pack.

- Reduce /tmp activity by setting the TMPDIR environment variable to $HOME/tmp in each user's .profile. This causes various utilities to put temporary files in the user's $HOME/tmp directory rather than in the common /tmp directory.

It is suggested that you place the temporary data in a separate file system. In this way, it will be easier to manage the space used by temporary files. How much temporary space is needed will depend on how z/OS UNIX is used. When the C compiler is invoked from the z/OS UNIX shell, the /tmp directory will be used for temporary data. Miscellaneous shell commands use the /tmp directory as well. If the z/OS UNIX shell is not installed, there will probably not be that much activity in the /tmp directory.

Another reason for placing the /tmp in a separate file system is to improve file system performance. If there is a lot of I/O activity on the /tmp directory, it would improve the performance if the OMVS.<SYSNAME>.TMP.HFS file system is placed on a different volume than the root file system.

7.11 Temporary file system

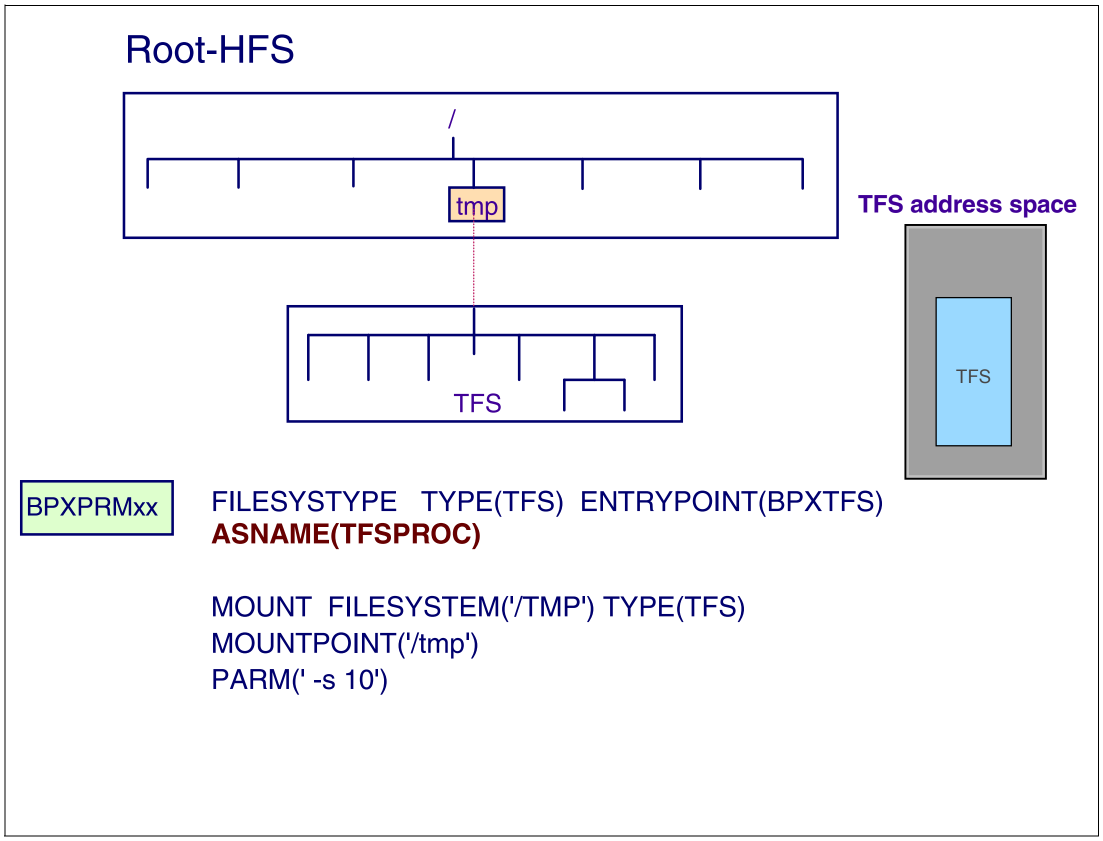


Figure 7-11 Temporary file system and temporary file system address space

The /tmp (temporary) directory is where many programs running on the system keep temporary copies of files or data. Users often use the tmp directory for working space knowing that they should not put anything out there that they want to keep. Currently the /tmp directory is part of the root HFS data set. The data in /tmp is not automatically deleted unless you have set up some mechanism (like the cron automation daemon) to delete the data at certain intervals. If the data is not deleted, you could run into space problems over time.

The TFS is an in-memory physical file system that supports in-storage mountable file systems. Because it is an in-memory file system, all data that is written to a TFS is lost over an IPL. This makes it a good candidate for the /tmp directory. Not only do you get a new clean /tmp file system at each IPL, you also get very high I/O access because it resides in a data space that is part of the kernel address space.

The ServerPac installation jobs already defined the following FILESYSTYPE definition in SYS1.PARMLIB(BPXPRMFS):

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

Unfortunately, you need an IPL to pick up any FILESYSTYPE definitions in BPXPRMFS. Because we have not IPLed the system since we installed the root HFS, the TFS physical file system is not started.

Edit SYS1.PARMLIB(BPXPRMFS), that is, add the mount statement to mount an in-memory file system at the /tmp mount point. You can add the following mount statement right under the FILESYSTYPE TYPE(TFS) definition:

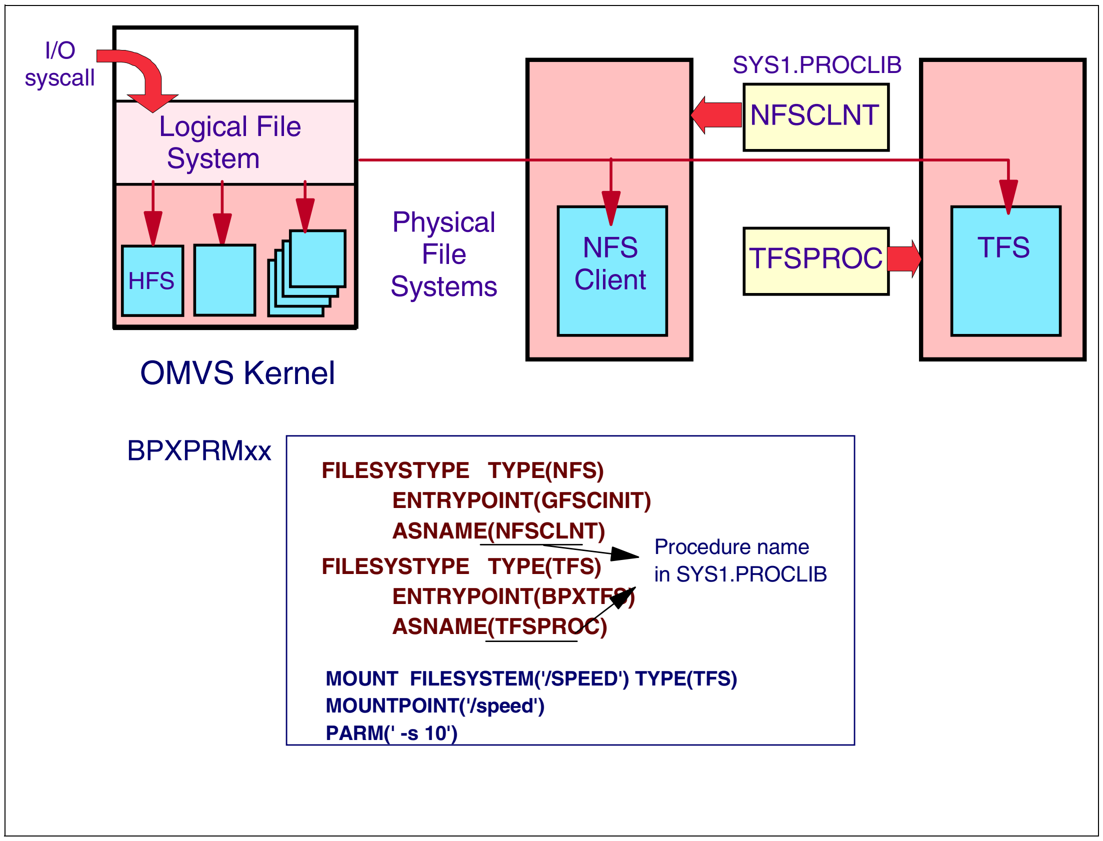
```
MOUNT FILESYSTEM('/TMP')
      MOUNTPOINT('/tmp')
      TYPE(TFS)
      PARM('-s 10')
```

(-s 10) allocates 10MB of storage.

Figure 7-11 shows the mount command for a TFS. Normally this would be included in BPXPRMxx. Key points to remember are the following:

- The file system name should be unique. If you use the pathname of the mount point, it will simplify the understanding of the output of the **`df`** command.
- Type must be a `TFS`.
- Mountpoint is `/tmp`.
- `Parm` specifies how much virtual storage the TFS should use. The default is 1 MB; `PARM (-s 10)` specifies 10 MB.
- You cannot mount a TFS using a DDname.
- If unmounted, all data stored in a TFS is lost; when remounted, the file system has only dot(.) and dot-dot(..) entries.

7.12 Colony address space



```
FILESYSTYPE   TYPE(NFS)
         ENTRYPOINT(GFSCINIT)
         ASNAME(NFSCLNT)
FILESYSTYPE   TYPE(TFS)
         ENTRYPOINT(BPXTFS)
         ASNAME(TFSPROC)

MOUNT  FILESYSTEM('/SPEED') TYPE(TFS)
MOUNTPOINT('/speed')
PARM(' -s 10')
```

Figure 7-12 Colony address spaces

The standard z/OS UNIX Physical File Systems (PFS) run as tasks in the kernel address space. However, it is possible to package a physical file system component in a separate address space, called a *colony address space*.

The Logical File System in the kernel provides the linkage to the new PFS in the colony address space. To a z/OS UNIX user application, there is no difference in interaction between accessing the standard PFS and one in a colony address space.

Running a PFS in a colony address space has some advantages, as follows:

- It is a simpler way for other components to add a new PFS to z/OS UNIX environment.
- A PFS can use a different security ID to kernel, and allocate its own z/OS data sets.
- A PFS can run as a standard process with normal C/C++ SYSCALL linkages.

When a PFS runs in a colony address space, an extra address space is created, and each PFS operation has a slightly longer pathlength.

Note: The cataloged procedure (named with ASNAME) must contain the statement:

```
EXEC PGM=BPXVCLNY
```

The following tasks must be done to run the colony address space:

- Define FILESYSTYPE TYPE(TFS) in BPXPRMxx.

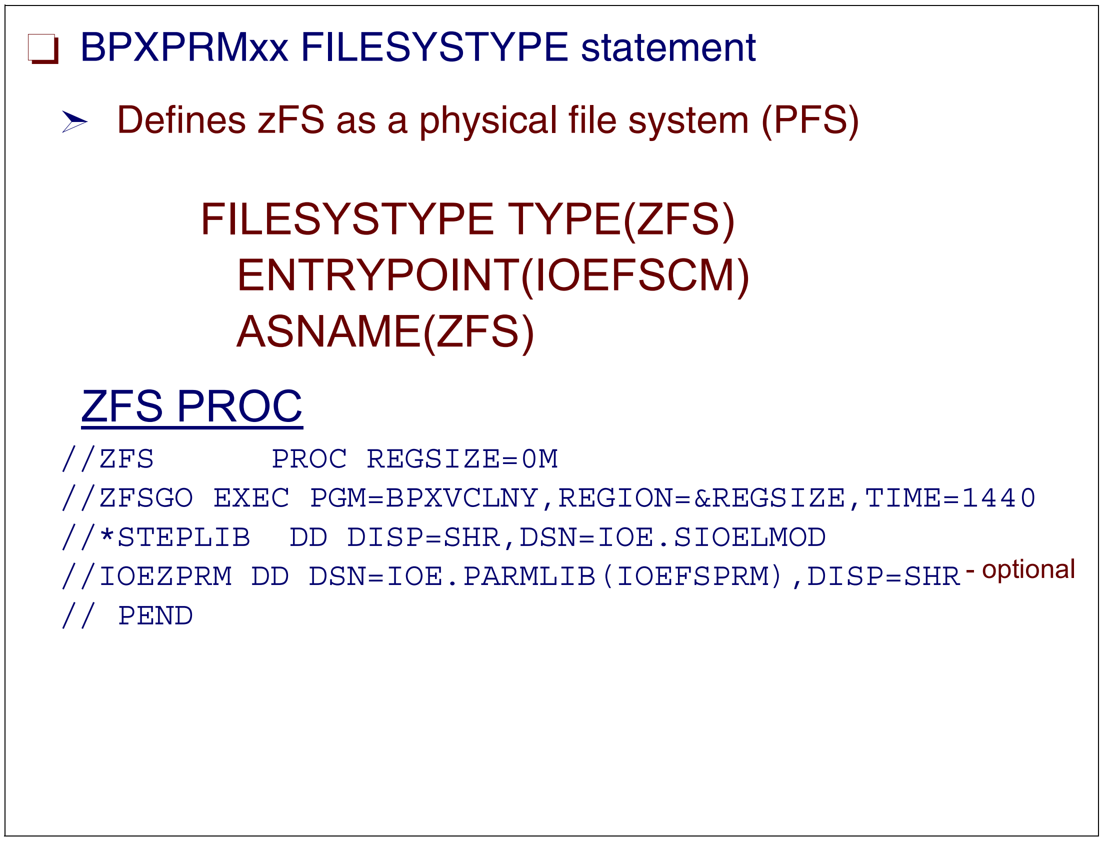
- Use a procedure name for the TFS, in the ASNAME in the BPXPRMxx.
- Define a profile in the started class for the procedure name.
- Add the MOUNT FILESYSTEM('/xxxx') to the BPXPRMxx.
- IPL the system.

The RACF profile for started class could look like:

```
RDEFINE STARTED TFSPROC.**   -
    STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
```

For physical file systems that do not run in the kernel address space, ASNAME specifies the name of the procedure in SYS1.PROCLIB used to start the address space in which this PFS will be initialized. The procedure name is also used as the name of the address space. For example, for the NFS client, you must create a procedure to run a PFS in a colony address space.

7.13 BPXPRMxx definitions for zFS



❑ BPXPRMxx FILESYSTYPE statement

➢ Defines zFS as a physical file system (PFS)

```
FILESYSTYPE TYPE(ZFS)
  ENTRYPOINT(IOEFSCM)
  ASNAME(ZFS)
```

ZFS PROC

```
//ZFS      PROC REGSIZE=0M
//ZFSGO EXEC PGM=BPXVCLNY,REGION=&REGSIZE,TIME=1440
//*STEPLIB  DD DISP=SHR,DSN=IOE.SIOELMOD
//IOEZPRM DD DSN=IOE.PARMLIB(IOEFSPRM),DISP=SHR - optional
// PEND
```

Figure 7-13 Colony address space definition for zFS

For zFS, you can specify the following statements in the z/OS UNIX BPXPRMxx member:

- Define the colony address space using the FILESYSTYPE statement.
- Beginning with z/OS V1R3, you can specify MOUNT statements to mount zFS file systems during the IPL or restart of OMVS.

A sample FILESYSTYPE statement is shown in Figure 7-13, and as follows:

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM) ASNAME(ZFS)
```

It includes an ASNAME parameter specifying the ZFS procedure to be started. It is required for PFSs that run in a colony address space.

zFS runs in a z/OS UNIX colony address space. A colony address space is an address space that is separate from the z/OS UNIX address space. HFS runs inside the z/OS UNIX address space.

Whether or not a PFS runs in a colony address space is controlled by the ASNAME parameter of the FILESYSTYPE statement for the PFS in the BPXPRMxx member of the SYS1.PARMLIB concatenation.

7.14 zFS colony address space

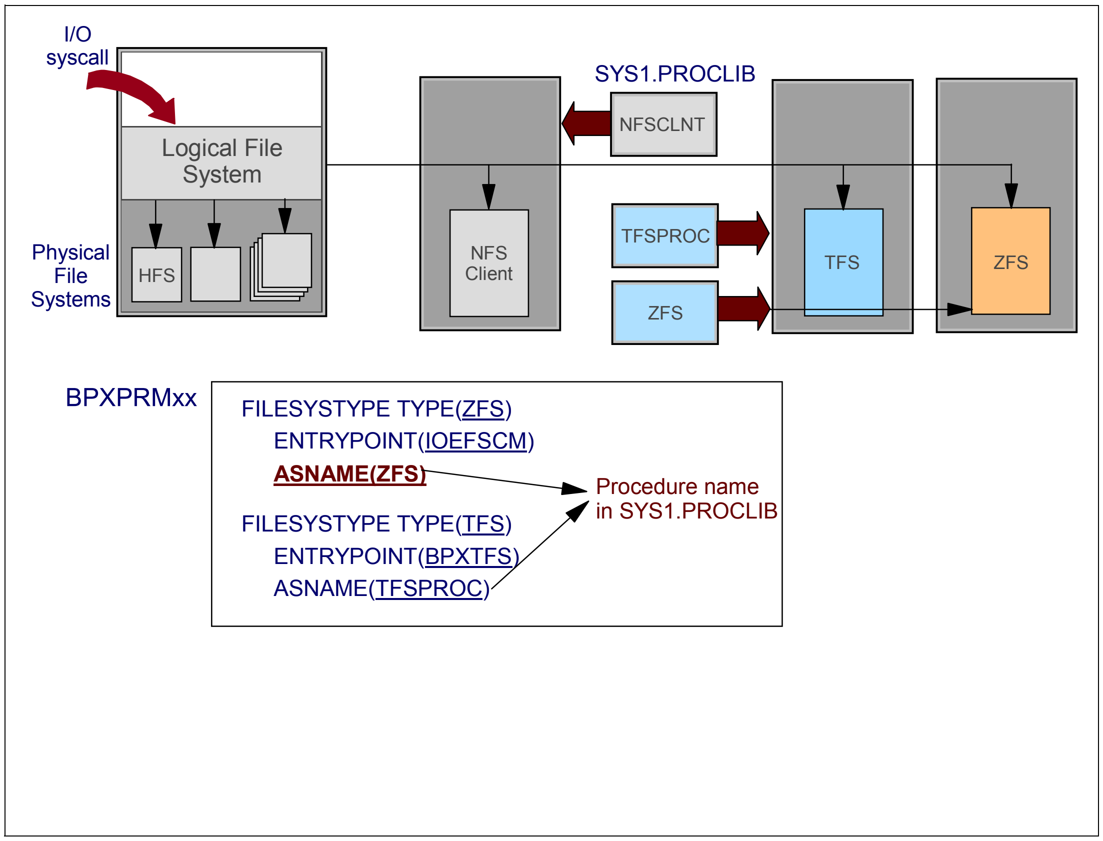


Figure 7-14 zFS colony address space

zFS runs in a UNIX System Services (USS) colony address space. A colony address space is an address space that is separate from the USS address space. HFS runs inside the USS address space and zFS runs in its own address space, as shown in the visual.

Here is a sample FILESYSTYPE statement for the zFS physical file system:

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM) ASNAME(ZFS)
```

It includes an ASNAME parameter specifying the zFS procedure to be started. It is required for PFSs that run in a colony address space.

Whether or not a PFS runs in a colony address space is controlled by the ASNAME parameter of the FILESYSTYPE statement for the PFS in the BPXPRMxx member of the SYS1.PARMLIB concatenation.

7.15 HFS data set

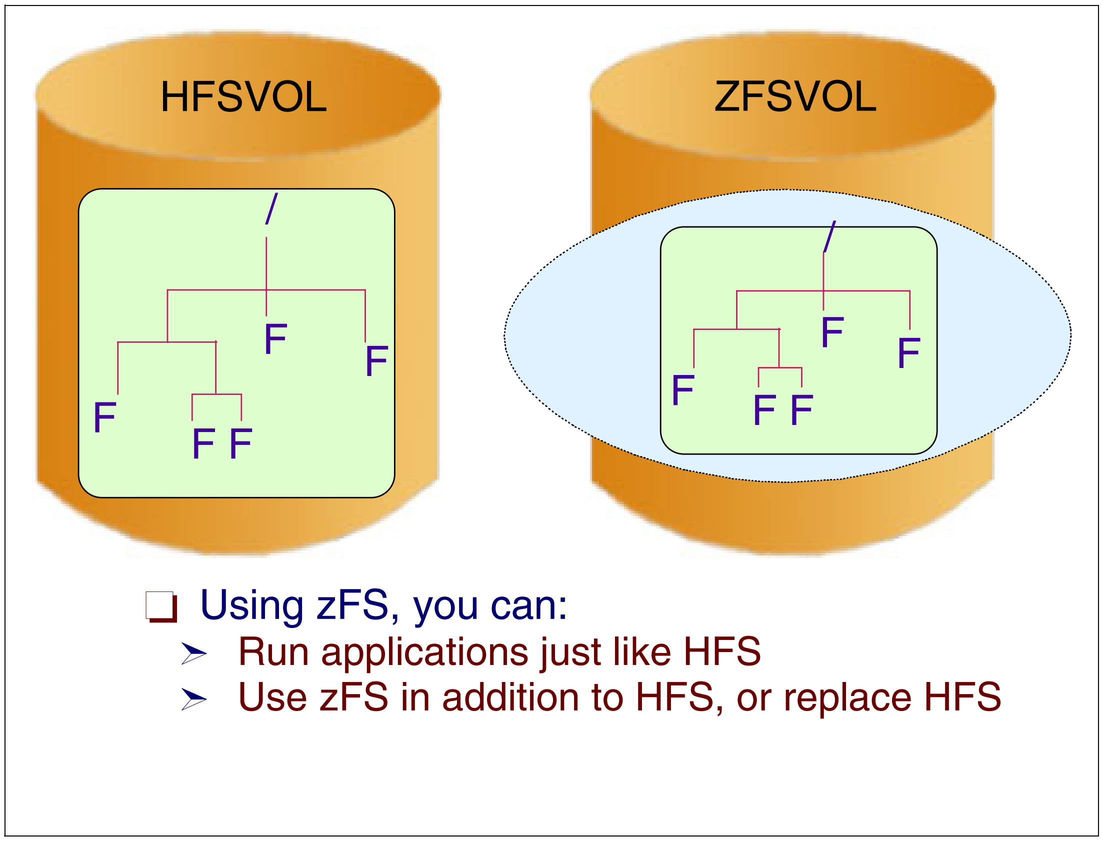

HFSVOL
ZFSVOL
/
F
F
F
F F
/
F
F
F
F F
❑ Using zFS, you can:
➢ Run applications just like HFS
➢ Use zFS in addition to HFS, or replace HFS

Figure 7-15 HFS data sets and zFS

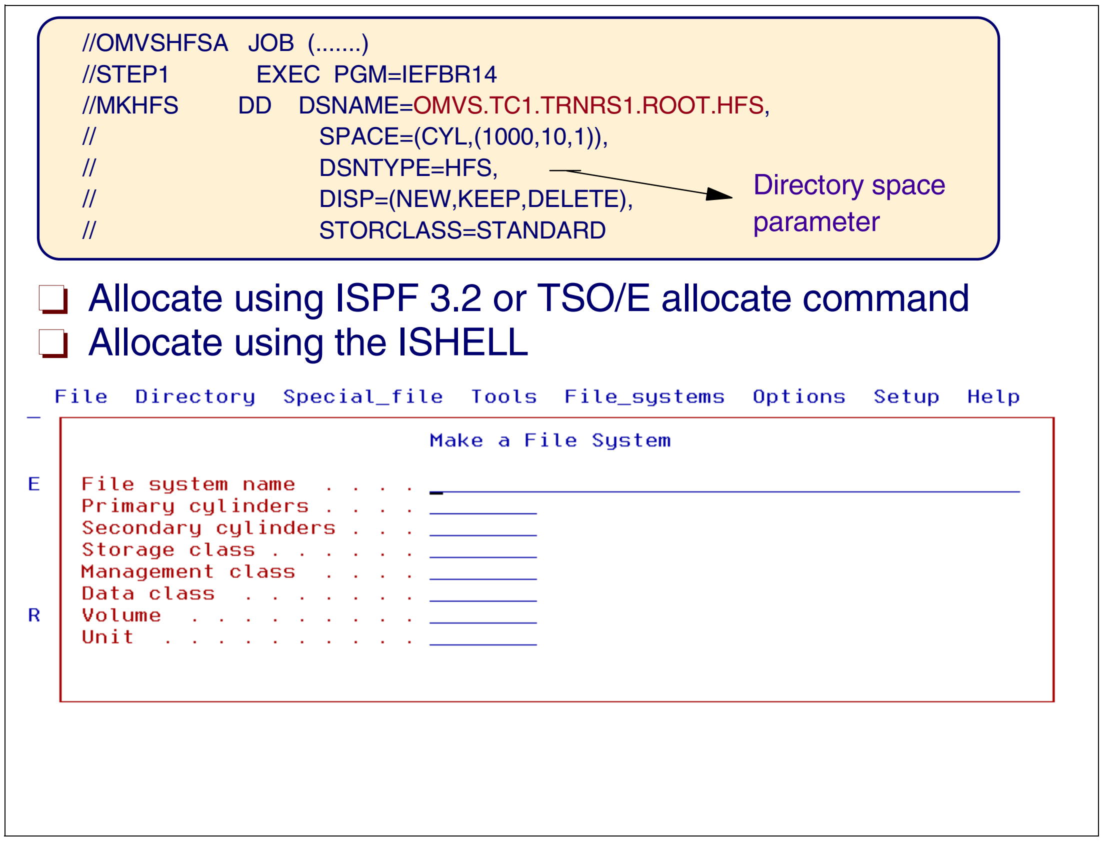
A z/OS UNIX hierarchical file system is contained in a data set type called HFS. An HFS data set must reside on an SMS-managed volume or a non SMS-managed volume. HFS data sets can reside with other z/OS data sets on SMS-managed volumes and non SMS-managed volumes. Multiple systems can share an HFS data set if it is mounted in read-only mode.

An HFS data set can have up to 123 extents, and the maximum size of the data set is one physical volume. For a 3390-Model 3, the maximum size is 2.838 GB. HFS data sets can only be accessed by z/OS UNIX.

The z/OS Distributed File Service (DFS) zSeries File System (zFS) is a z/OS UNIX file system that can be used in addition to the HFS. zFS provides significant performance gains in accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to that of HFS.

zFS file systems contain files and directories that can be accessed with the z/OS hierarchical file system application programming interfaces on the z/OS operating system.

7.16 Allocate the HFS file system



```
//OMVSHFSA   JOB  (.......)
//STEP1           EXEC  PGM=IEFBR14
//MKHFS        DD    DSNAME=OMVS.TC1.TRNRS1.ROOT.HFS,
//                      SPACE=(CYL,(1000,10,1)),
//                      DSNTYPE=HFS,
//                      DISP=(NEW,KEEP,DELETE),
//                      STORCLASS=STANDARD
```

- Allocate using ISPF 3.2 or TSO/E allocate command
- Allocate using the ISHELL

```
   File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 _
                          Make a File System

 E    File system name  . . . . _
      Primary cylinders . . . . ________
      Secondary cylinders . . . ________
      Storage class . . . . . . ________
      Management class  . . . . ________
      Data class  . . . . . . . ________
 R    Volume  . . . . . . . . . ________
      Unit  . . . . . . . . . . ________
```

Figure 7-16 How to allocate an HFS data set

Before z/OS UNIX can be started, an HFS data set for the root file system must be allocated. This can be done by a batch job, as shown in Figure 7-16, or by using the TSO/E allocate command or ISPF.

Note: The directory space parameter is not used for HFS data sets, but it must be specified as a non-zero number.

The following example shows how to use the TSO/E ALLOCATE command to allocate the same data set as shown on the figure:

```
alloc da('OMVS.<SYSNAME>.[<SYSR1>].prod_name.HFS') dsntype(HFS)
 space(40,10) dsorg(PO) cyl storclass(STANDARD)
```

You can also use the ISHELL to allocate a new HFS data set, if you are familiar with the shell, by selecting:

TSO ISHELL → File_systems → NEW

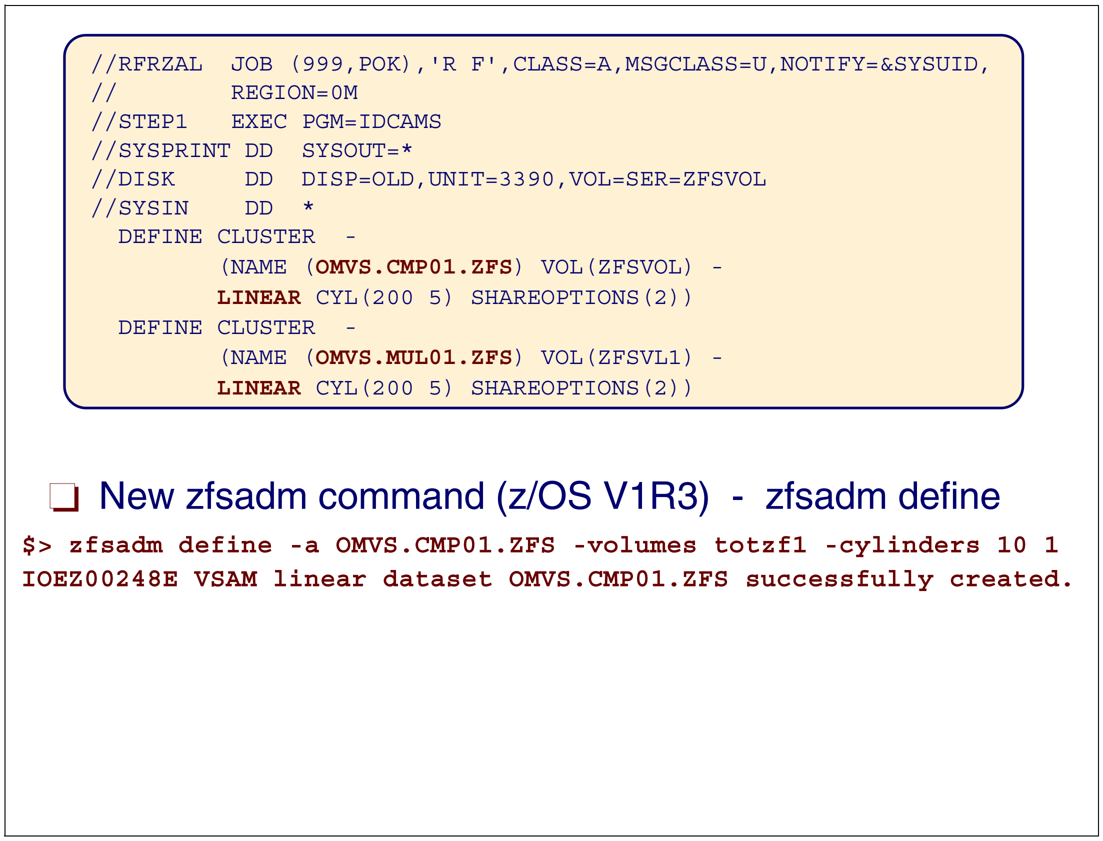
The name of the Root file system can be anything an installation decides. In this example, it is:

```
OMVS.<SYSNAME>.[<SYSR1>].prod_name.HFS
```

Practice shows that it is useful to include the system name in the HFS name.

You should design a file system structure of mounted file systems in a way that there are few changes to the root file system. This is similar to the recommendation for the z/OS master catalog. We discuss suggestions for the file system structure in more detail in later sections. However, the size to allocate for the root file system depends on this structure. Assuming that users will have their data in mounted user file systems, the root file system does not need to be very large.

7.17 Allocate zFS Aggregates



```
//RFRZAL  JOB (999,POK),'R F',CLASS=A,MSGCLASS=U,NOTIFY=&SYSUID,
//        REGION=0M
//STEP1   EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//DISK     DD  DISP=OLD,UNIT=3390,VOL=SER=ZFSVOL
//SYSIN    DD  *
  DEFINE CLUSTER  -
         (NAME (OMVS.CMP01.ZFS) VOL(ZFSVOL) -
         LINEAR CYL(200 5) SHAREOPTIONS(2))
  DEFINE CLUSTER  -
         (NAME (OMVS.MUL01.ZFS) VOL(ZFSVL1) -
         LINEAR CYL(200 5) SHAREOPTIONS(2))
```

- New zfsadm command (z/OS V1R3) - zfsadm define

```
$> zfsadm define -a OMVS.CMP01.ZFS -volumes totzf1 -cylinders 10 1
IOEZ00248E VSAM linear dataset OMVS.CMP01.ZFS successfully created.
```

Figure 7-17 How to allocate a zFS aggregate

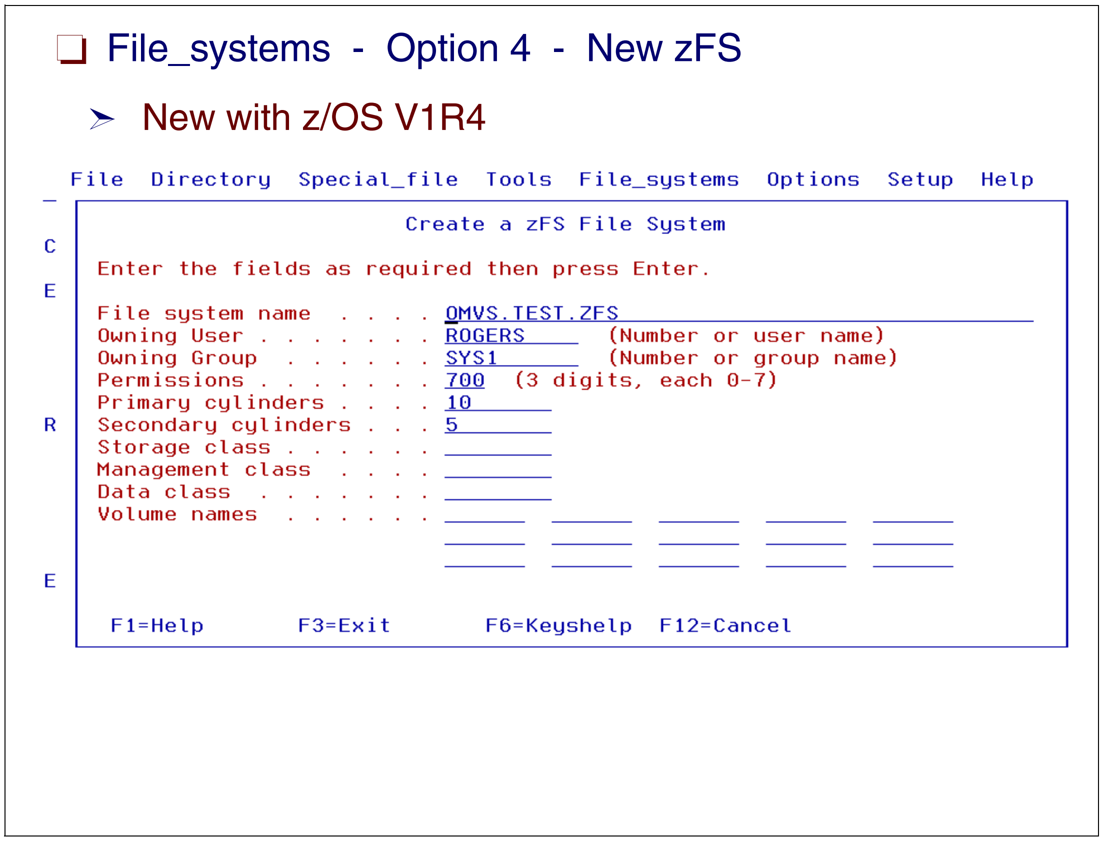
A zFS aggregate is created by defining a VSAM linear data set (LDS) and then formatting that VSAM LDS as an aggregate. This is done once for each zFS aggregate. You cannot assign more than one VSAM LDS per aggregate. An aggregate can contain one or more zFS file systems. A zFS file system is equivalent to an HFS file system.

The VSAM LDS is allocated with the VSAM utility program IDCAMS, as shown in Figure 7-17. The JCL shows the allocation of both types of aggregates.

The **`zfsadm define`** command defines a VSAM LDS. The VSAM LDS is available to be formatted as a zFS aggregate. The command creates a DEFINE CLUSTER command string for a VSAM LDS with SHAREOPTIONS(2) and passes it to the IDCAMS utility. If a failure occurs, the **`zfsadm define`** command may display additional messages from IDCAMS indicating the reason for the failure.

Note: The issuer of the **`zfsadm define`** command requires sufficient authority to create the VSAM LDS.

7.18 Allocate the aggregate from ISHELL


- File_systems - Option 4 - New zFS
  - New with z/OS V1R4

```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
_ ┌──────────────────────────────────────────────────────────────────────────┐
  │                         Create a zFS File System                         │
C │                                                                          │
  │ Enter the fields as required then press Enter.                           │
E │                                                                          │
  │ File system name . . . . QMVS.TEST.ZFS____________________________       │
  │ Owning User . . . . . . . ROGERS____  (Number or user name)              │
  │ Owning Group . . . . . . SYS1______  (Number or group name)              │
  │ Permissions . . . . . . . 700  (3 digits, each 0-7)                      │
  │ Primary cylinders . . . . 10_______                                      │
R │ Secondary cylinders . . . 5________                                      │
  │ Storage class . . . . . . _________                                      │
  │ Management class . . . . _________                                       │
  │ Data class . . . . . . . _________                                       │
  │ Volume names . . . . . . ______ ______ ______ ______ ______              │
  │                          ______ ______ ______ ______ ______              │
  │                          ______ ______ ______ ______ ______              │
E │                                                                          │
  │                                                                          │
  │  F1=Help       F3=Exit       F6=Keyshelp  F12=Cancel                     │
  └──────────────────────────────────────────────────────────────────────────┘
```

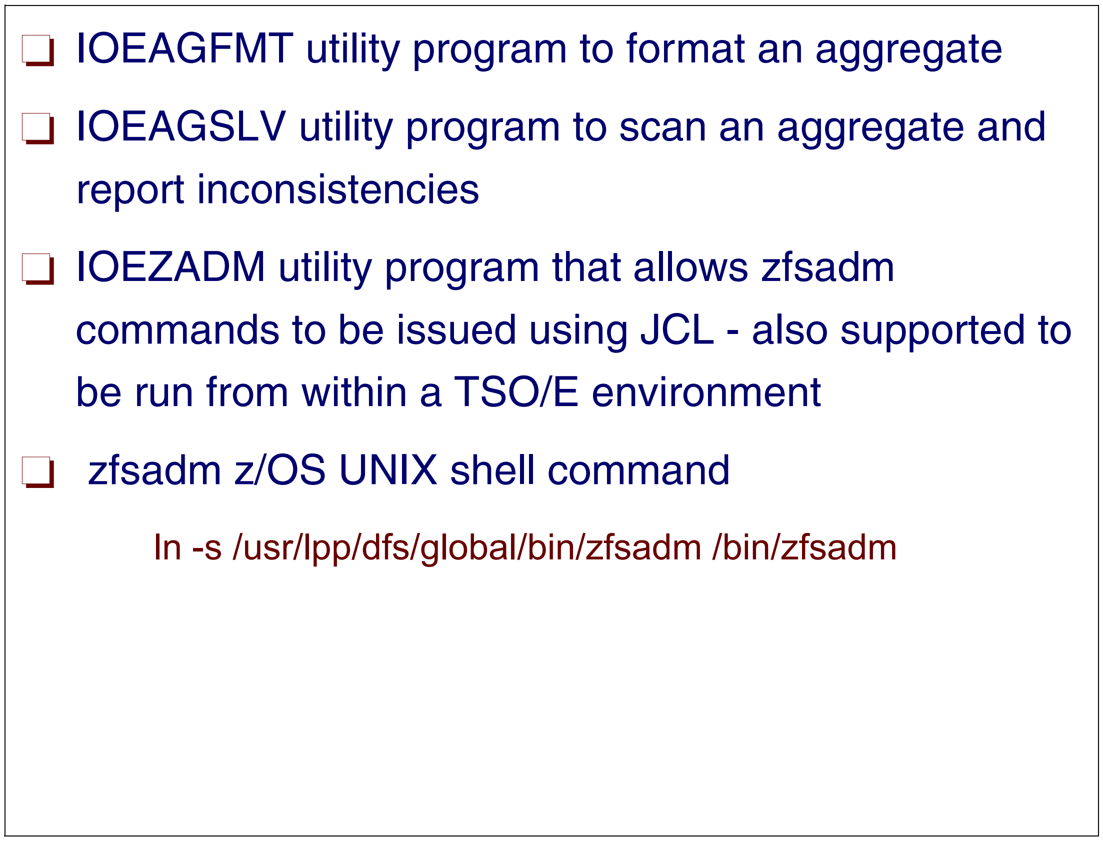
Figure 7-18 How to allocate a zFS aggregate from the ISHELL

The panel shown in Figure 7-18 is new with z/OS V1R4. It allows you to allocate a zFS aggregate from the ISHELL.

Once you enter the ISHELL, place the cursor under File_systems and press Enter. Then select Option 4 (New zFS) and the panel shown in the figure appears. Type in the information to create the aggregate. You now a a linear VSAM data set.

The aggregate must be formatted before it can be used for zFS file systems. During the format, you specify whether it is to be a compatibility mode or multi-file mode aggregate.

7.19 zFS utilities and commands



- IOEAGFMT utility program to format an aggregate
- IOEAGSLV utility program to scan an aggregate and report inconsistencies
- IOEZADM utility program that allows zfsadm commands to be issued using JCL - also supported to be run from within a TSO/E environment
- zfsadm z/OS UNIX shell command

  ln -s /usr/lpp/dfs/global/bin/zfsadm /bin/zfsadm

Figure 7-19 zFS utilities and commands

The following utility programs are provided:

IOEAGFMT	Used to format an aggregate.
IOEAGSLV	Used to scan an aggregate and report inconsistencies.
IOEZADM	Allows **`zfsadm`** commands to be issued using JCL. It is also supported to be run in a TSO/E environment.

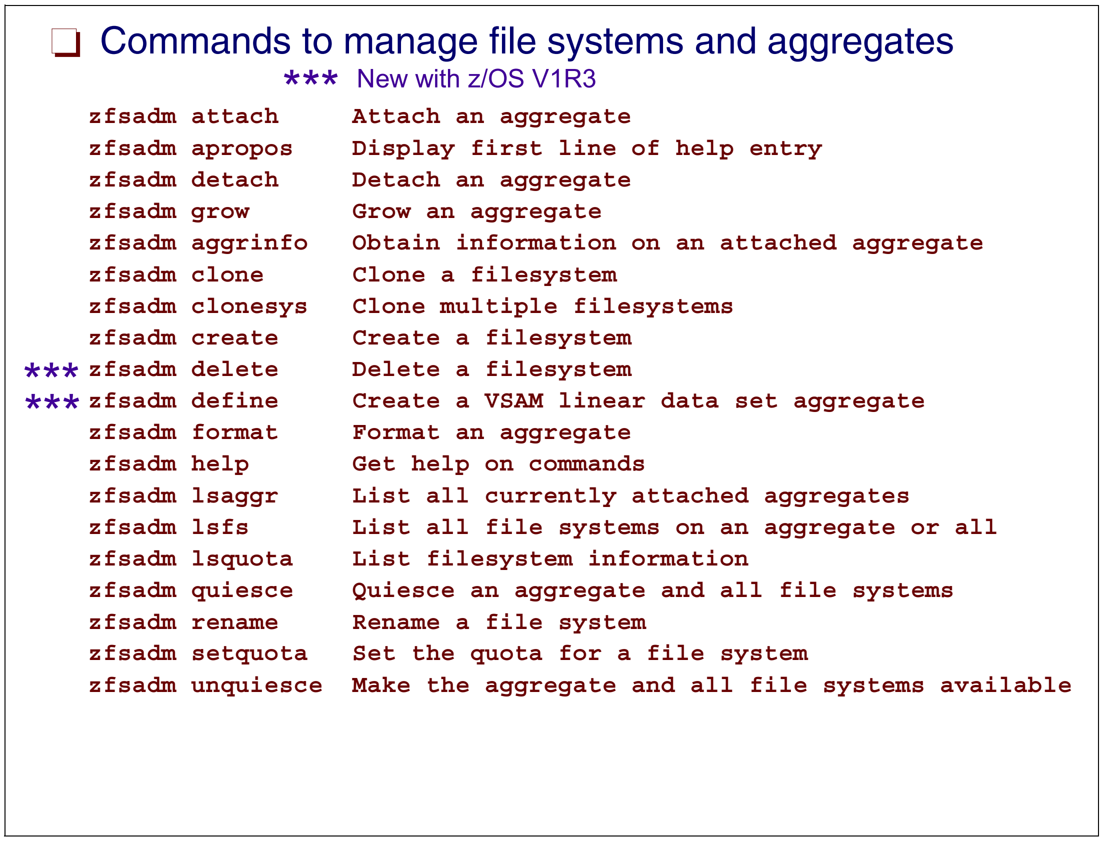
Note: Although zfsadm and IOEZADM are physically different modules, they contain identical code. Whenever we reference zfsadm as a zFS administration command in this book, we mean both **`zfsadm`** and IOEZADM. Therefore, IOEZADM can be used as a TSO/E command; it performs the same functions as the **`zfsadm`** command.

zFS provides a recovery mechanism that uses a zFS file system log to verify or correct the structure of an aggregate. This recovery mechanism is invoked by an operator command, **`ioeagslv`**.

Verify that during your z/OS Distributed File Service installation a symbolic link was created for access to the **`zfsadm`** commands. If for some reason this was not done, issue the following command to create a symbolic link in the /bin directory:

```
  #> ln -s /usr/lpp/dfs/global/bin/zfsadm /bin/zfsadm
```

7.20 `zfsadm` command



❑ Commands to manage file systems and aggregates

*** New with z/OS V1R3

	zfsadm attach	Attach an aggregate
	zfsadm apropos	Display first line of help entry
	zfsadm detach	Detach an aggregate
	zfsadm grow	Grow an aggregate
	zfsadm aggrinfo	Obtain information on an attached aggregate
	zfsadm clone	Clone a filesystem
	zfsadm clonesys	Clone multiple filesystems
	zfsadm create	Create a filesystem
***	zfsadm delete	Delete a filesystem
***	zfsadm define	Create a VSAM linear data set aggregate
	zfsadm format	Format an aggregate
	zfsadm help	Get help on commands
	zfsadm lsaggr	List all currently attached aggregates
	zfsadm lsfs	List all file systems on an aggregate or all
	zfsadm lsquota	List filesystem information
	zfsadm quiesce	Quiesce an aggregate and all file systems
	zfsadm rename	Rename a file system
	zfsadm setquota	Set the quota for a file system
	zfsadm unquiesce	Make the aggregate and all file systems available

Figure 7-20 The **`zfsadm`** *command and its subcommands*

zFS provides utility programs and z/OS UNIX commands to assist in the customization of the aggregates and file systems. These utilities and commands are to be used by system administrators.

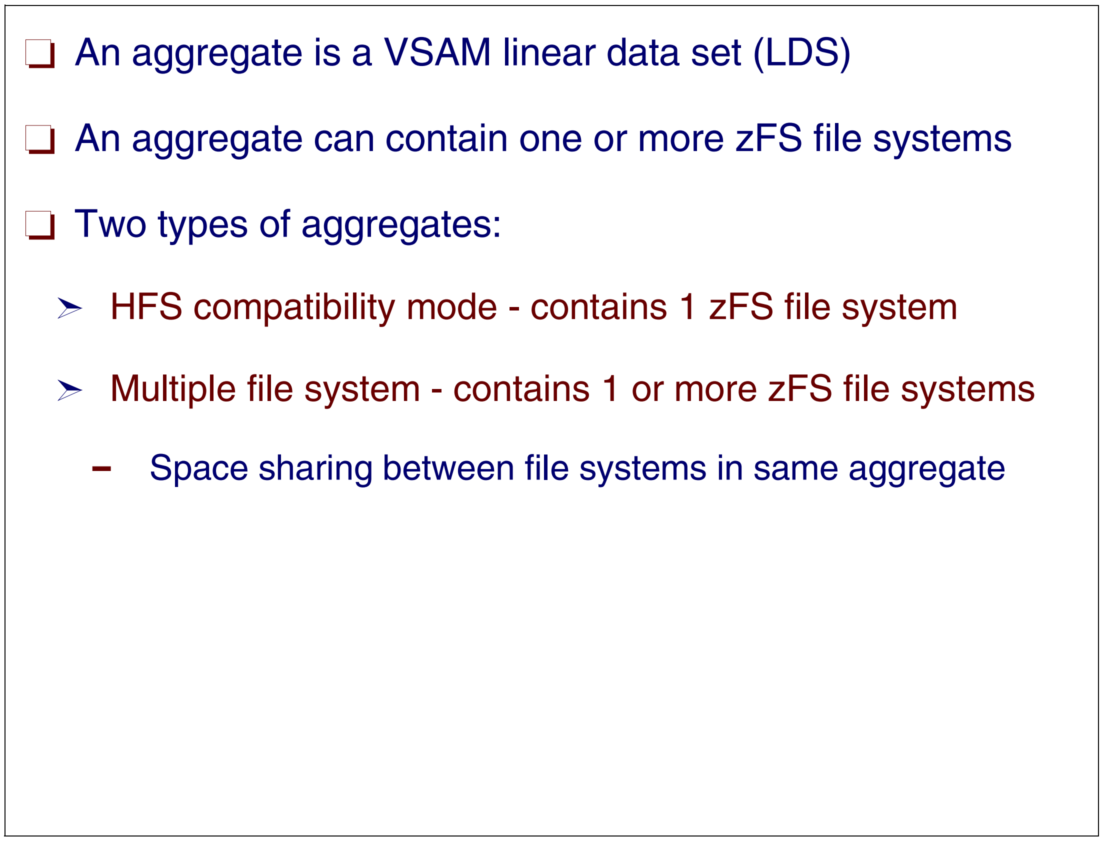
The **`zfsadm`** command and the IOEZADM utility program can be used to manage file systems and aggregates.

The **`zfsadm`** command can be run as a UNIX shell command from:

- A z/OS UNIX system services shell (OMVS) or a (z/OS UNIX) telnet session
- A batch job using the BPXBATCH utility program
- TSO foreground or in batch mode using z/OS UNIX APIs (SYSCALL commands, Callable Services)

zFS file systems can be created using JCL, or by using the **`zfsadm`** command. Figure 7-20 shows the **`zfsadm`** command with its subcommands.

7.21 zFS aggregates



- ❑ An aggregate is a VSAM linear data set (LDS)
- ❑ An aggregate can contain one or more zFS file systems
- ❑ Two types of aggregates:
  - ➢ HFS compatibility mode - contains 1 zFS file system
  - ➢ Multiple file system - contains 1 or more zFS file systems
    - – Space sharing between file systems in same aggregate

Figure 7-21 zFS aggregate descriptions

A zFS aggregate is a data set that contains zFS file systems. The aggregate is a VSAM LDS and is a container that can contain one or more zFS file systems. An aggregate can only have one VSAM LDS, but it can contain an unlimited number of file systems. The name of the aggregate is the same as the VSAM LDS name.

Sufficient space must be available on the volume or volumes, as multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during any extension of a primary allocation. VSAM LDSs greater than 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set.

After the aggregate is created, formatting of the aggregate is necessary before any file systems can exist in it. A zFS file system is a named entity that resides in a zFS aggregate. It contains a root directory and can be mounted into the UNIX System Services hierarchy. While the term file system is not a new term, a zFS file system resides in a zFS aggregate, which is different from an HFS file system.

zFS aggregates come in two types:

- ► Compatibility mode aggregates
- ► Multi-file system aggregates

7.22 zFS compatibility mode aggregate

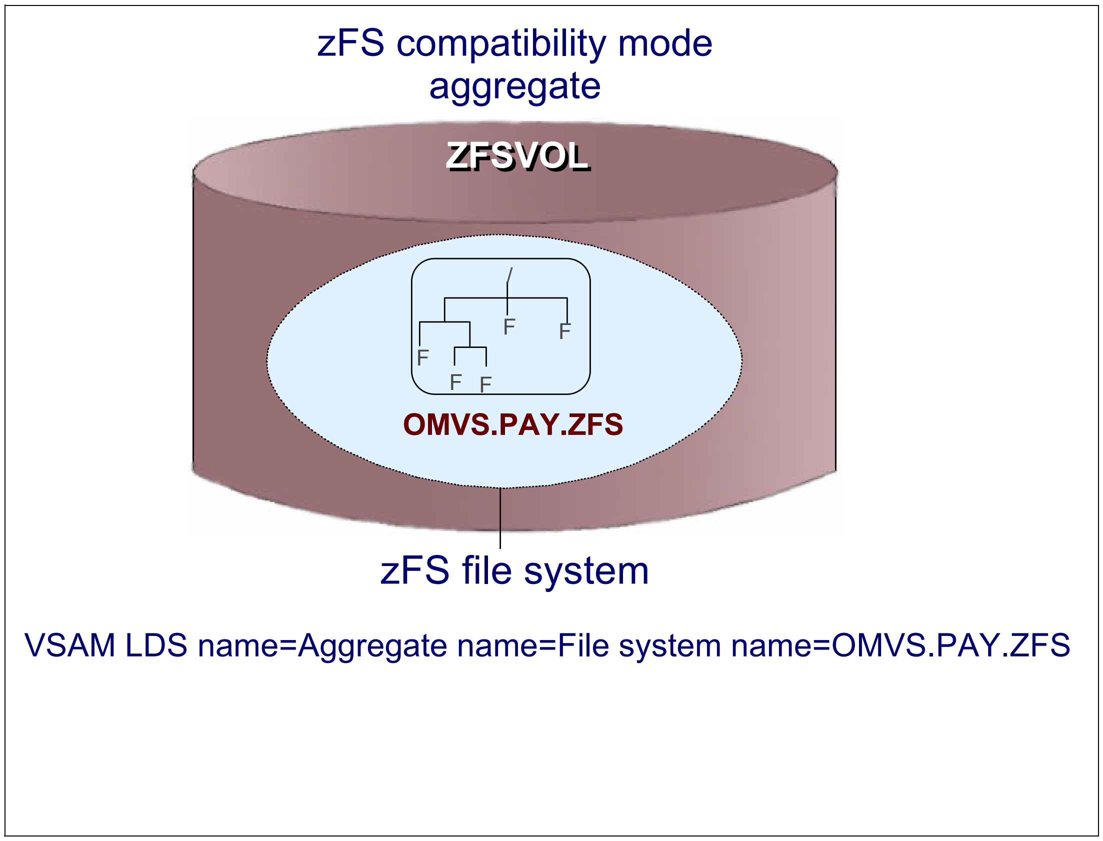

zFS compatibility mode aggregate
ZFSVOL
/
F
F
F
F
F
OMVS.PAY.ZFS
zFS file system
VSAM LDS name=Aggregate name=File system name=OMVS.PAY.ZFS

Figure 7-22 zFS compatibility mode aggregate

A compatibility mode aggregate can contain only one zFS file system, making this type of aggregate more like an HFS file system. This is flagged in the aggregate when it is created. The name of the file system is the same as the name of the aggregate, which is the same as the VSAM LDS cluster name. The file system size (called a *quota*) in a compatibility mode aggregate is set to the size of the aggregate. Compatibility mode aggregates are VSAM linear data sets instead of HFS data sets.

We recommend that you start using compatibility mode aggregates first, since they are similar to the familiar HFS data sets.

7.23 Multiple file mode aggregate

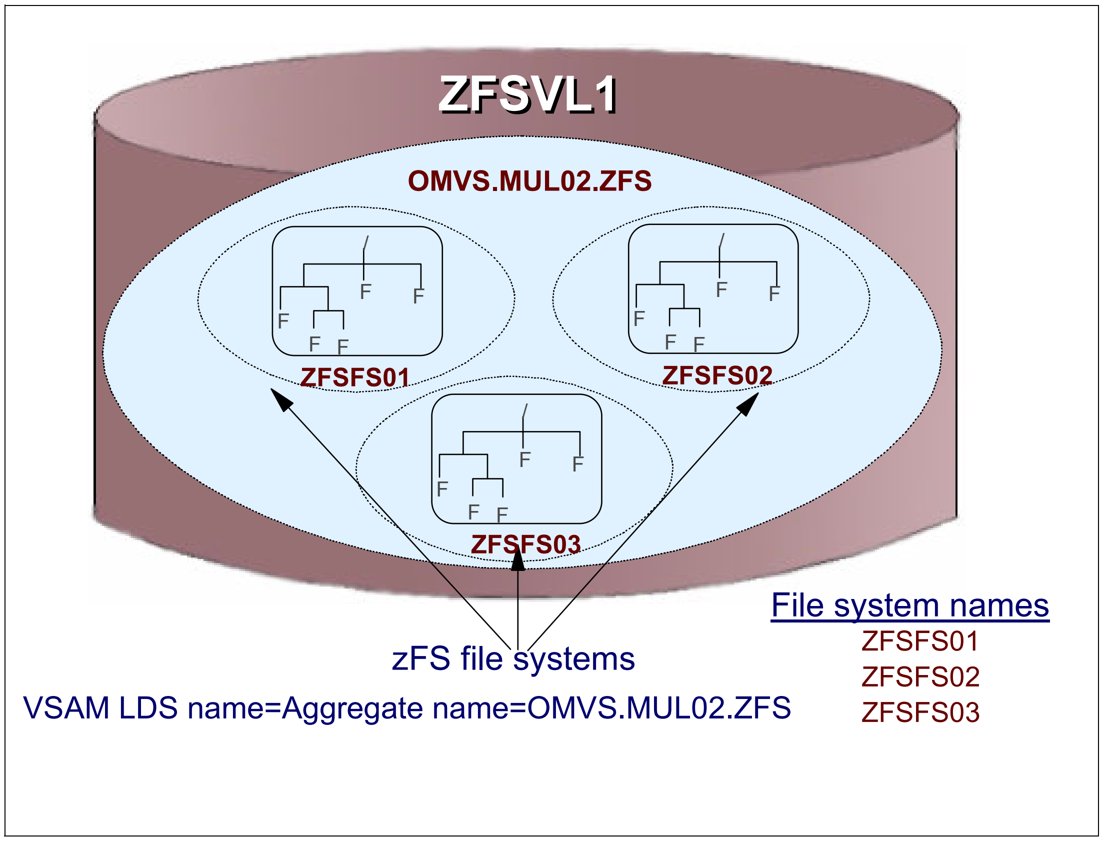


Figure 7-23 zFS multiple file mode aggregate

A multi-file mode aggregate allows the administrator to define multiple zFS file systems in a single aggregate. This type of aggregate can contain one or more zFS file systems. This allows space that becomes available when files are deleted in one file system to be made available to other file systems in the same aggregate data set or space sharing.

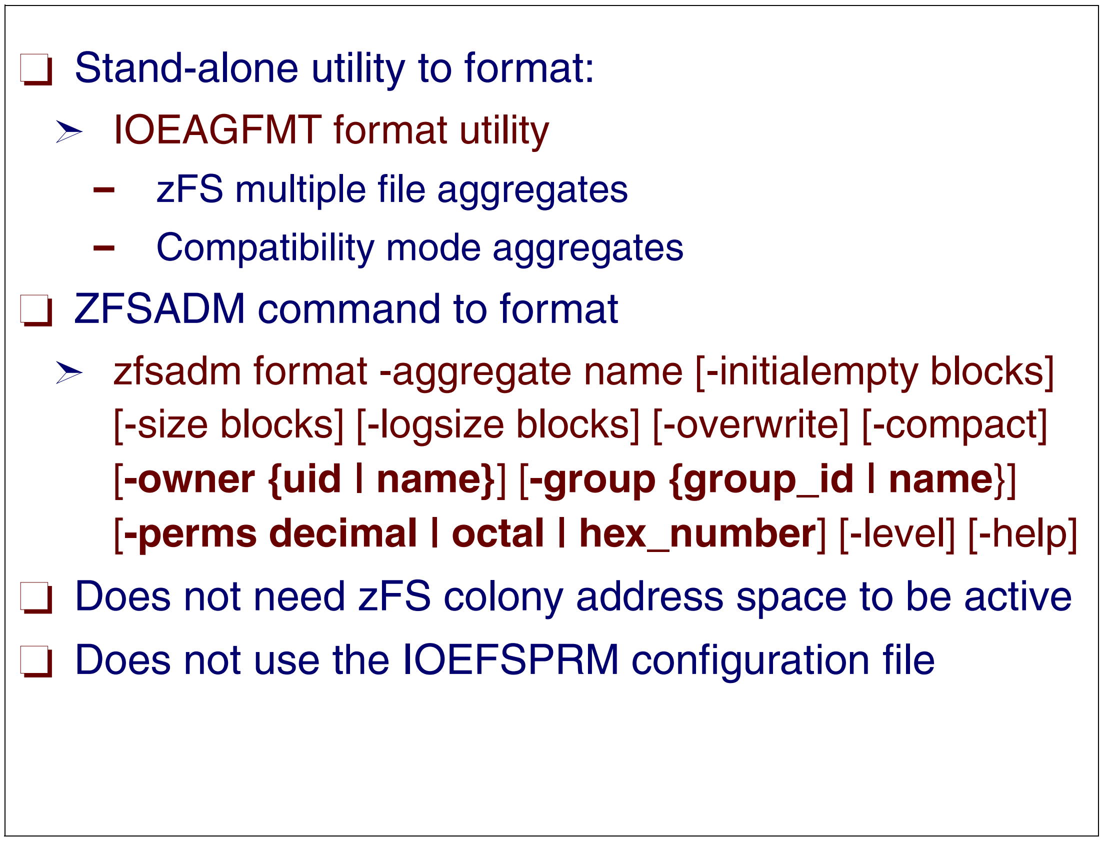
Space sharing

Space sharing means that if you have multiple file systems in a single data set, and files are removed from one of the file systems—which frees DASD space—another file system can use that space when new files are created. This new type of file system is called a multi-file system aggregate.

The multi-file system aggregate OMVS.MUL02.ZFS, shown in Figure 7-23, can contain multiple zFS file systems. This makes it possible to do space sharing between the zFS file systems within the aggregate.

The multiple file system aggregate has its own name. This name is assigned when the aggregate is created. It is always the same as the VSAM LDS cluster name. Each zFS file system in the aggregate has its own file system name. This name is assigned when the particular file system in the aggregate is created. Each zFS file system also has a predefined maximum size, called the quota.

7.24 Format the zFS aggregate



- Stand-alone utility to format:
  - IOEAGFMT format utility
    - zFS multiple file aggregates
    - Compatibility mode aggregates
- ZFSADM command to format
  - zfsadm format -aggregate name [-initialempty blocks] [-size blocks] [-logsize blocks] [-overwrite] [-compact] [**-owner {uid | name}**] [**-group {group_id | name}**] [**-perms decimal | octal | hex_number**] [-level] [-help]
- Does not need zFS colony address space to be active
- Does not use the IOEFSPRM configuration file

Figure 7-24 Formatting a zFS aggregate

The VSAM linear data set must be formatted to be used as a zFS aggregate. Two options are available for formatting an aggregate:

- The IOEAGFMT format utility
- The **`zfsadm`** command

Both options use the same parameters, as follows:

```
zfsadm format -aggregate name [-initialempty blocks] [-size blocks] [-logsize blocks]
[-overwrite] [-compat] [-owner {uid | name}] [-group {group_id | name}]
[-perms decimal | octal | hex_number] [-level] [-help]
```

This utility formats the primary allocation and, if requested by using the `-size` parameter (with a value greater than the primary allocation), a single *extension*. An extension is a single call to the Media Manager to extend the data set to the size specified in the `-size` parameter. It is completely independent of the number of cylinders specified in the secondary allocation when the data set was defined (the secondary allocation could have been 0).

If your initial VSAM LDS allocation is for multiple volumes, the initial formatting of the zFS aggregate is limited to the primary allocation (in this case, the first volume), and one extension (in this case, the second volume). After that, you must use multiple **`zfsadm grow`** commands to grow to each new volume.

7.25 Format the aggregates

```
A compatibility mode aggregate is formatted with this JCL:

//RFRAGF   JOB (999,POK),'R F',CLASS=A,MSGCLASS=U,NOTIFY=&SYSUID,
// REGION=0M
//STEP1    EXEC PGM=IOEAGFMT,
//           PARM=(' -aggregate omvs.cmp01.zfs -compat ')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
A multi-file system aggregate is formatted with this JCL:
//RFRAGF   JOB (999,POK),'R F',CLASS=A,MSGCLASS=U,NOTIFY=&SYSUID,
// REGION=0M
//STEP1    EXEC PGM=IOEAGFMT,
//           PARM=(' -aggregate omvs.mul01.zfs ')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
```

Figure 7-25 Examples of formatting a zFS aggregate

Figure 7-25 shows the JCL for formatting compatibility mode and multi-file mode aggregates.

The sample JCL was run using all the default values, shown underlined in the following table.

Parameter	Values	Description
-aggregate	VSAM LDS cluster name	The name of the data set to format.
-size	4K, 8K, 16K, 32K, 64K	The size in bytes of the logical block.
-owner	uid | name	Specifies the owner for the root directory of the file system for compat mode only. Default is uid of issuer of `ioeagfmt`.
-group	gid | name	Specifies the group owner for the root directory for compat mode only. Default is gid of issuer.
-logsize	a number	The size in blocks of the log. The default is 1% of the aggrsize.
-overwrite	NA	Reformat an existing aggregate.
-compat	NA	Create a compatibility mode aggregate.
-perms	o755	Permission bit settings for root directory.

7.26 IOEAGFMT successful messages

```
IOEZ00004I Loading dataset 'omvs.cmp01.zfs'.
IOEZ00005I Dataset 'omvs.cmp01.zfs' loaded successfully.
*** Using default initialempty value of 1.
*** Using default number of (8192-byte) blocks: 17999
*** Defaulting to 179 log blocks(maximum of 19 concurrent transactions).
Done.  /dev/lfs1/omvs.cmp01.zfs is now an zFS aggregate.
IOEZ00071I Attaching aggregate OMVS.CMP01.ZFS to create hfs-compatible file system
IOEZ00074I Creating file system of size 140487K, owner id 0, group id 2, permissions x1ED
IOEZ00048I Detaching aggregate OMVS.CMP01.ZFS
IOEZ00077I HFS-compatibility aggregate OMVS.CMP01.ZFS has been successfully created


IOEZ00004I Loading dataset 'omvs.mul01.zfs'.
IOEZ00005I Dataset 'omvs.mul01.zfs' loaded successfully.
*** Using default initialempty value of 1.
*** Using default number of (8192-byte) blocks: 17999
*** Defaulting to 179 log blocks(maximum of 19 concurrent transactions).
Done.  /dev/lfs1/omvs.mul01.zfs is now an zFS aggregate.
```

Figure 7-26 zFS formatting messages

When you format a compatibility mode aggregate (output of the JCL shown in the figure), the following processing is done:

- The aggregate is attached, allowing creation of a file system.
- A file system is created for the aggregate with the same name as the aggregate.

When you format a multi-file mode aggregate, an attach is not done and no file system is created. These two steps must be done separately.

7.27 Defining IEOFSPRM options

Table 7-1 zFS IOEFSPRM member options

Option	Values	Description
adm_threads	n, <u>10</u>	The number of threads to handle pfsctl or mount requests.
auto_attach	<u>ON</u>, OFF	Whether aggregates in IOEFSPRM are automatically attached at start-up.
user_cache_size	n, <u>256M</u>	The size of the cache used to contain file data.
meta_cache_size	n, <u>32M</u>	The size of the cache used to contain metadata.
log_cache_size	n, <u>64M</u>	The size of the cache used to contain log buffers.
sync_interval	n, <u>30</u>	The number of seconds between syncs.
vnode_cache_size	n, <u>8192</u>	The initial size of the internal zFS vnode cache.
nbs	ON, <u>OFF</u>	New Block Security. A global specification of whether we guarantee that new block data that has not made it to disk before a crash, will be shown as binary zeros after the crash.
aggrfull	(n,m), <u>OFF</u>	The threshold and increment for reporting aggregate full msgs to the op.
fsfull	(n,m), <u>OFF</u>	The threshold and increment for reporting file system full msgs to the op.

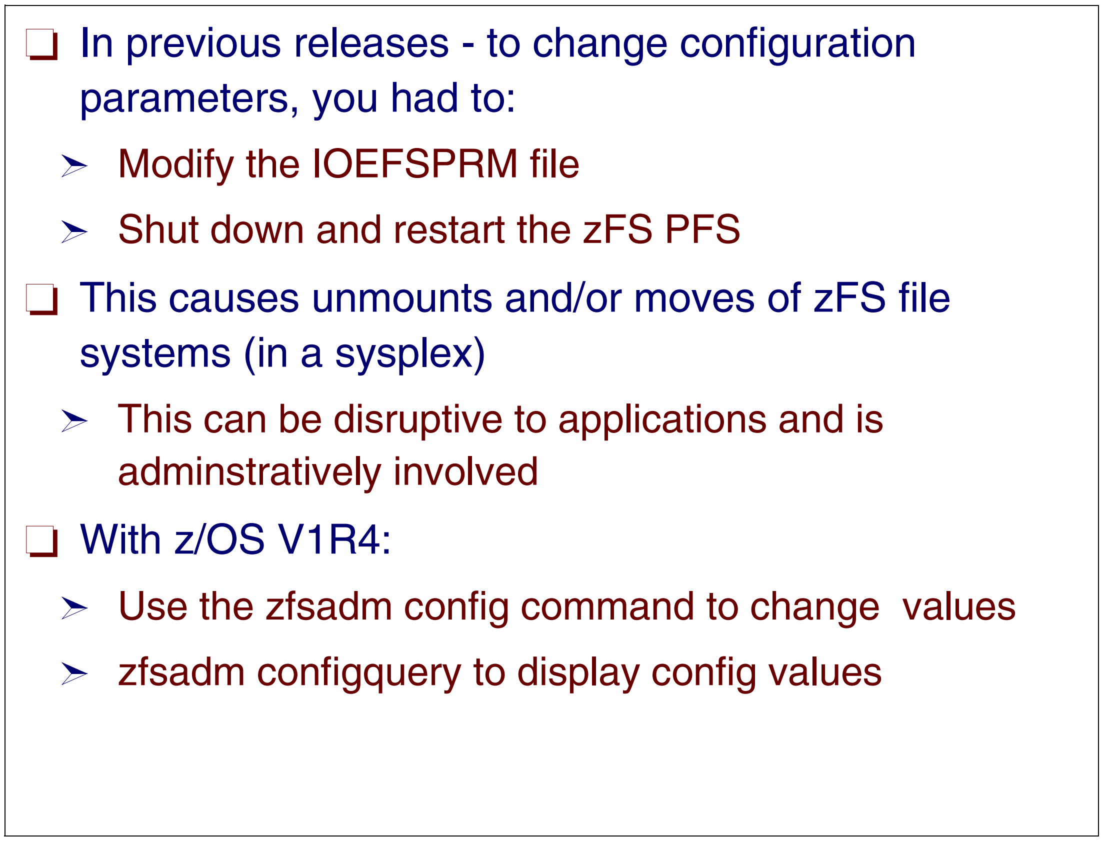
The best and only reasonable option is to define IOEFSPRM as a member of a PDS data set. This allows updates while the zFS PFS is active.

Table 7-1 lists the processing options for the zFS PFS and the definitions of the multi-file system aggregates. There is no mandatory information in this file, therefore it is not required. The options all have defaults. Aggregates can all be compatibility mode aggregates (which do not need definitions). Multi-file system aggregates can be attached by using the **`zfsadm attach`** command. They do not need definitions in IOEFSPRM to be attached using **`zfsadm attach`**. However, if you need to specify any options (for tuning purposes, for example) or if you want to have any multi-file system aggregates automatically attached when zFS is started, you need to have an IOEFSPRM file.

The location of the IOEFSPRM file is specified by the IOEZPRM DD statement in the ZFS PROC. The IOEFSPRM file is normally a PDS member, so the IOEZPRM DD might look like the following:

```
//IOEZPRM  DD  DSN=SYS4.PVT.PARMLIB(IOEFSPRM),DISP=SHR
```

7.28 Dynamic configuration: z/OS V1R4



- ❑ In previous releases - to change configuration parameters, you had to:
  - ➢ Modify the IOEFSPRM file
  - ➢ Shut down and restart the zFS PFS
- ❑ This causes unmounts and/or moves of zFS file systems (in a sysplex)
  - ➢ This can be disruptive to applications and is adminstratively involved
- ❑ With z/OS V1R4:
  - ➢ Use the zfsadm config command to change values
  - ➢ zfsadm configquery to display config values

Figure 7-28 Changing the zFS IOEFSPRM member parameters dynamically

zFS has a IOEFSPRM configuration file that specifies the processing options for the zFS PFS and some definitions for multi-file aggregates. Before z/OS V1R4, to change any configuration file parameters, you had to do the following:

1. Modify the IOEFSPRM file.
2. Shut down and restart the zFS PFS.

Doing this causes unmounts and potential moves of zFS file systems in a sysplex environment, which could be disruptive to applications and is administratively involved.

Two new **`zfsadm`** commands have been added with z/OS V1R4; they are used to change configuration values dynamically without a shutdown, restart the zFS PFS, and display the current value of the zFS configuration options.

The **`zfsadm config`** command changes the value of zFS configuration options in memory that were specified in the IOEFSPRM file (or defaulted).

The **`zfsadm configquery`** command displays the current value of zFS configuration options retrieved from the zFS address space memory, rather than from the IOEFSPRM file.

7.29 IOEFSPRM files options

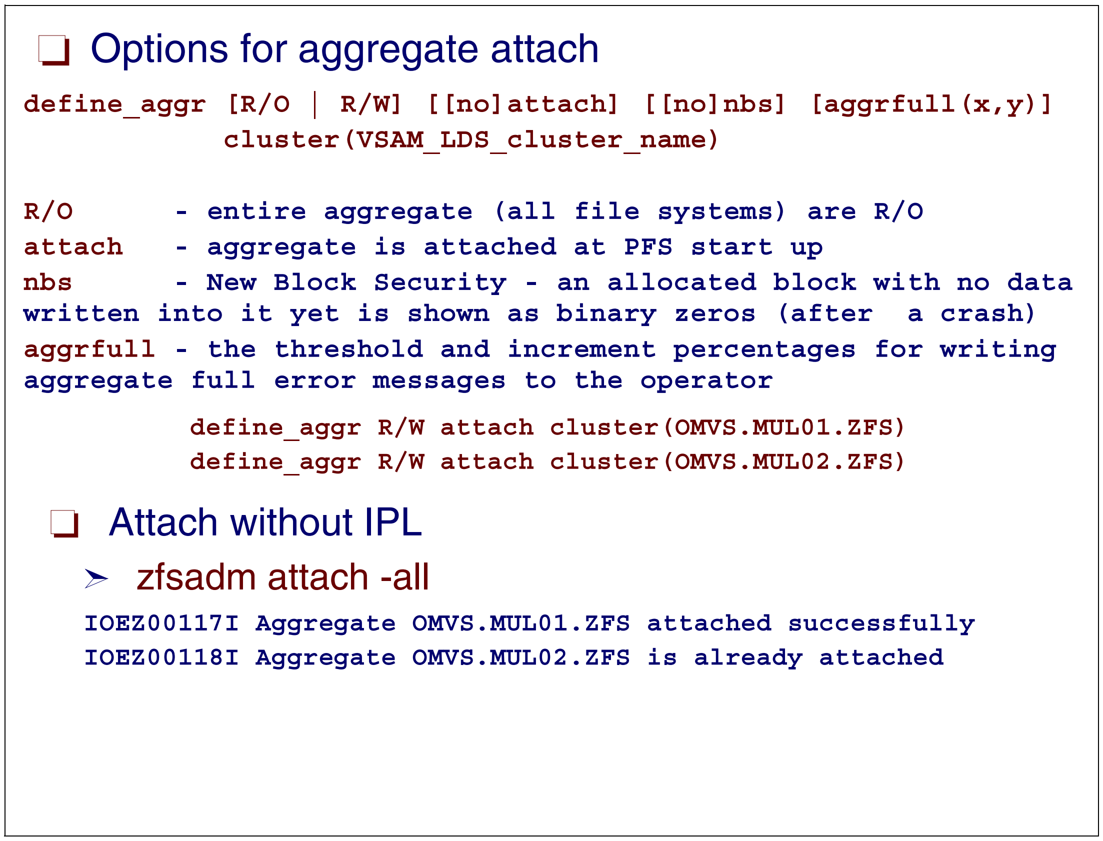
```
zfsadm config [-admin_threads number]
[-user_cache_size number]
[-meta_cache_size number]
[-log_cache_size number]
[-sync_interval number]
[-vnode_cache_size number]
[-nbs {on|off}]
[-fsfull threshold,increment]
[-aggrfull threshold,increment]
[-trace_dsnPDSE_dataset_name]
[-tran_cache_size number]
[-msg_output_dsn Seq_dataset_name]
[-user_cache_readahead {on|off}]
[-metaback_cache_size number]
[-fsgrow increment,times]
[-aggrgrow {on|off}]
[-allow_dup_fs {on|off}]
[-level]
[-help]
```

Figure 7-29 The IOEFSPRM member parameters

The **zfsadm config** command changes the configuration options (in memory) that were specified in the IOEFSPRM file (or defaulted). The IOEFSPRM file is not changed. If you want the configuration specification to be permanent, you need to modify the IOEFSPRM file since ZFS reads the IOEFSPRM file to determine the configuration values the next time ZFS is started. The values that can be specified for each option are the same as the values that can be specified for that option in the IOEFSPRM file.

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be root or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

7.30 IOEFSPRM definitions for attach



❑ Options for aggregate attach

```
define_aggr [R/O | R/W] [[no]attach] [[no]nbs] [aggrfull(x,y)]
            cluster(VSAM_LDS_cluster_name)

R/O      - entire aggregate (all file systems) are R/O
attach   - aggregate is attached at PFS start up
nbs      - New Block Security - an allocated block with no data
written into it yet is shown as binary zeros (after  a crash)
aggrfull - the threshold and increment percentages for writing
aggregate full error messages to the operator
          define_aggr R/W attach cluster(OMVS.MUL01.ZFS)
          define_aggr R/W attach cluster(OMVS.MUL02.ZFS)
```

❑ Attach without IPL

- zfsadm attach -all

```
IOEZ00117I Aggregate OMVS.MUL01.ZFS attached successfully
IOEZ00118I Aggregate OMVS.MUL02.ZFS is already attached
```

Figure 7-30 Methods of attaching multiple file mode aggregates

The attach of the aggregate is an operation that attaches a multi-file system aggregate to a system. This makes the aggregate and all its file systems known to the zFS Physical File System running on that system.

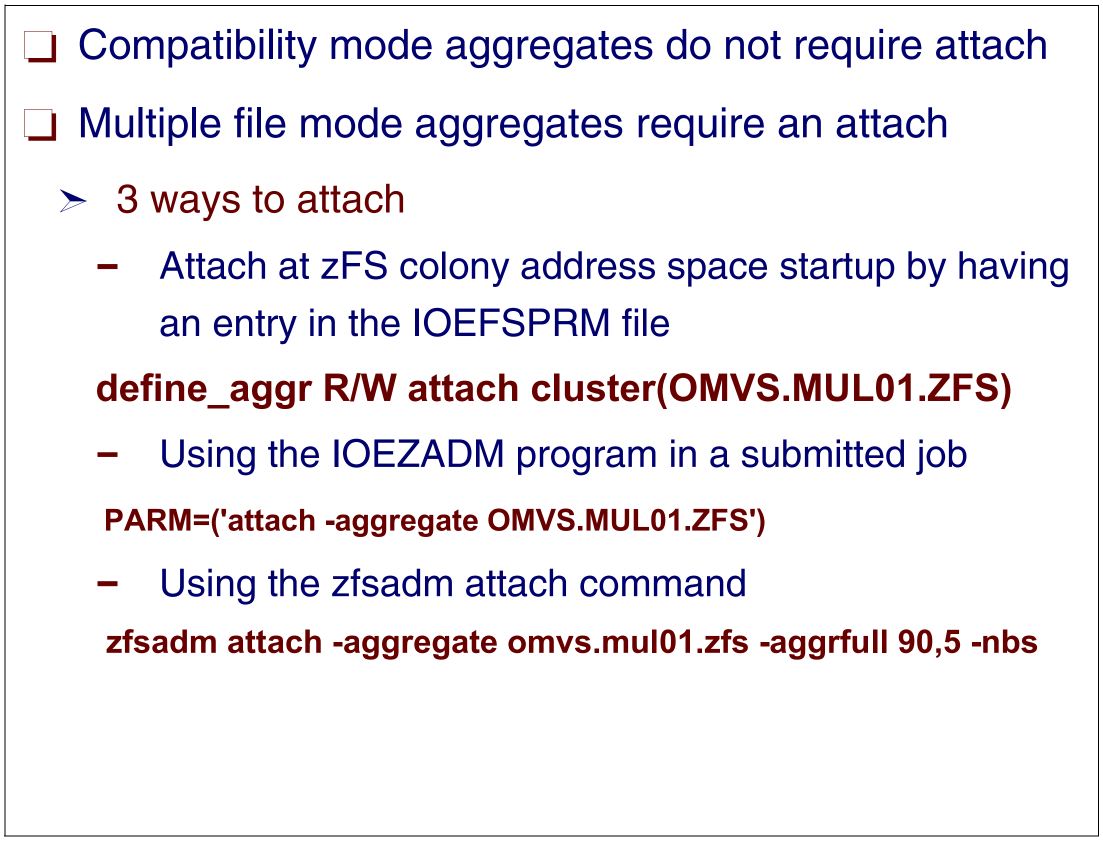
If you created and formatted a zFS multi-file system aggregate, you can add an entry in the IOEFSPRM member for the aggregate. This causes the multi-file system aggregate to be attached when zFS is started. Add the following lines to the IOEFSPRM member:

```
define_aggr R/W attach cluster(OMVS.MUL01.ZFS)
define_aggr R/W attach cluster(OMVS.MUL02.ZFS)
```

If you add those statements while zFS is already running, you can use the command shown in the figure to attach the aggregates immediately without having to do an IPL.

```
zfsadm attach -all
IOEZ00117I Aggregate OMVS.MUL01.ZFS attached successfully
IOEZ00118I Aggregate OMVS.MUL02.ZFS is already attached
```

7.31 Attaching a multi-file mode aggregate



- ❑ Compatibility mode aggregates do not require attach
- ❑ Multiple file mode aggregates require an attach
  - ➢ 3 ways to attach
    - – Attach at zFS colony address space startup by having an entry in the IOEFSPRM file

      define_aggr R/W attach cluster(OMVS.MUL01.ZFS)
    - – Using the IOEZADM program in a submitted job

      PARM=('attach -aggregate OMVS.MUL01.ZFS')
    - – Using the zfsadm attach command

      zfsadm attach -aggregate omvs.mul01.zfs -aggrfull 90,5 -nbs

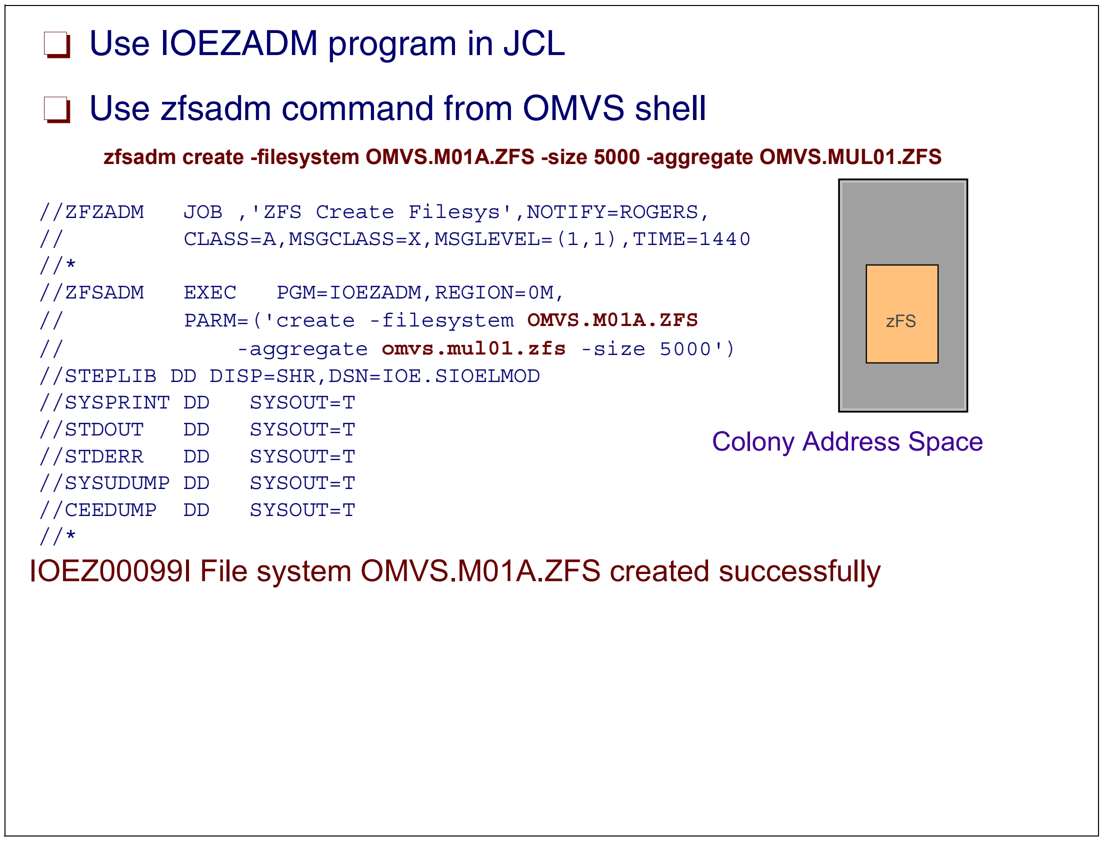
Figure 7-31 Attaching multi-file mode aggregates

If a zFS multiple file system aggregate is created after zFS is started, you must attach the zFS aggregate to tell the zFS PFS about it. Use the IOEFSPRM file, the **`zfsadm`** command, or use JCL to attach an aggregate, as follows:

```
//ZFSATTA JOB ,'ZFS Attach Aggregate',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//ZFSADMA  EXEC   PGM=IOEZADM,REGION=0M,
// PARM=('attach -aggregate OMVS.MUL02.ZFS')
//*STEPLIB DD DISP=SHR,DSN=hlq.SIOELMOD
//SYSPRINT DD   SYSOUT=T
//STDOUT   DD   SYSOUT=T
//STDERR   DD   SYSOUT=T
//SYSUDUMP DD   SYSOUT=T
//CEEDUMP  DD   SYSOUT=T
//*
```

A sample of the JCL to run PGM=IOEZADM is supplied in IOE.SIOESAMP(IOEZADM). You can also use **`IOEZADM`** from the TSO foreground as a command.

7.32 Defining multi-fIle zFS file systems



- ❑ Use IOEZADM program in JCL
- ❑ Use zfsadm command from OMVS shell

zfsadm create -filesystem OMVS.M01A.ZFS -size 5000 -aggregate OMVS.MUL01.ZFS

```
//ZFZADM   JOB ,'ZFS Create Filesys',NOTIFY=ROGERS,
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
//*
//ZFSADM   EXEC   PGM=IOEZADM,REGION=0M,
//         PARM=('create -filesystem OMVS.M01A.ZFS
//             -aggregate omvs.mul01.zfs -size 5000')
//STEPLIB DD DISP=SHR,DSN=IOE.SIOELMOD
//SYSPRINT DD   SYSOUT=T
//STDOUT   DD   SYSOUT=T
//STDERR   DD   SYSOUT=T
//SYSUDUMP DD   SYSOUT=T
//CEEDUMP  DD   SYSOUT=T
//*
```

IOEZ00099I File system OMVS.M01A.ZFS created successfully

Figure 7-32 Defining zFS file systems for multi-file mode aggregates

Once an aggregate has been attached, a zFS file system can be created in the aggregate. The zFS colony address space must be running. There are three methods to define a zFS file system, as follows:

- Use the IOEZADM utility program using JCL.
- Use the **IOEZADM** command from TSO.
- Use the **zfsadm** command from the OMVS shell.

Attention: The user must have superuser authority to use the IOEZADM program or the proper authority for the **zfsadm** command.

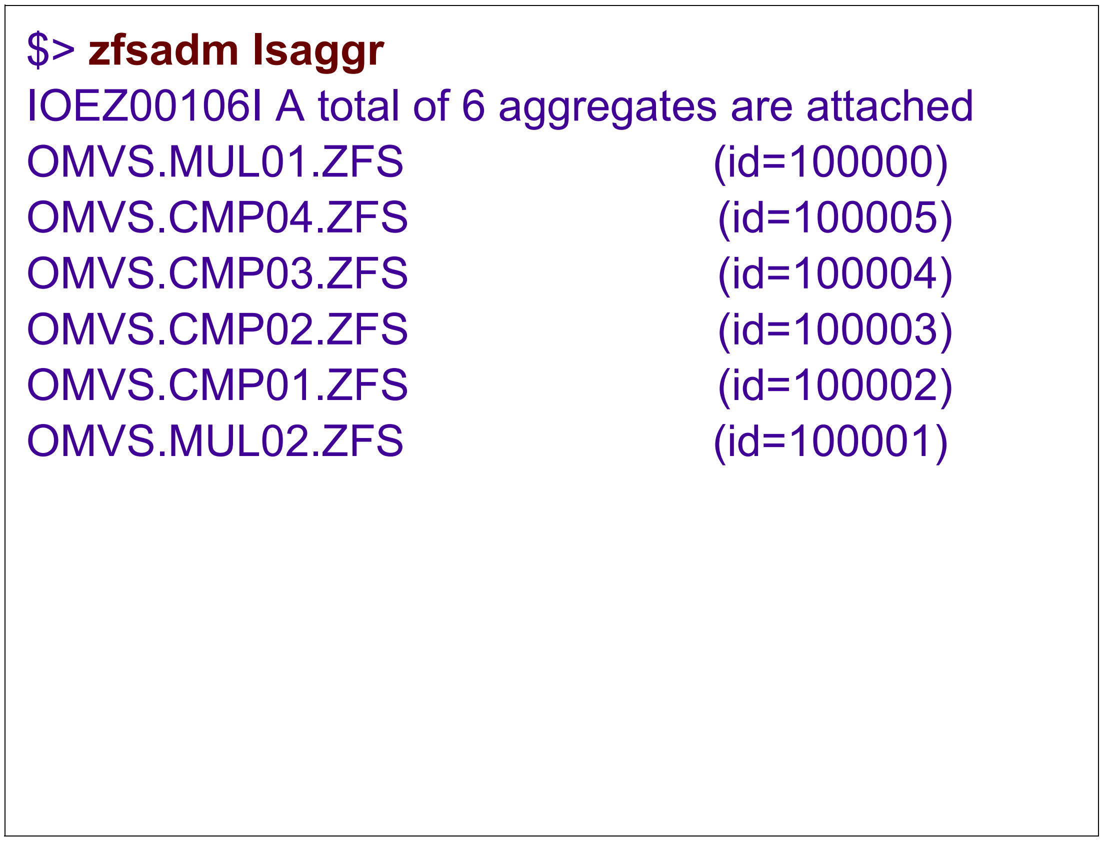
The IOEZADM program using JCL

To define a zFS file system in an aggregate, use the IOEZADM program with the subcommand CREATE. When defining the parm= keyword, as shown in Figure 7-32, specify the following:

- The file system name, which is case sensitive
- The aggregate name where the file system is to be defined
- The size of the file system in 1K blocks, which is a required parameter

Note: There is no default logical size for a zFS file system. The zFS file system name is case sensitive, and must be uppercase.

7.33 Display attached aggregates



```
$> zfsadm lsaggr
IOEZ00106I A total of 6 aggregates are attached
OMVS.MUL01.ZFS                              (id=100000)
OMVS.CMP04.ZFS                              (id=100005)
OMVS.CMP03.ZFS                              (id=100004)
OMVS.CMP02.ZFS                              (id=100003)
OMVS.CMP01.ZFS                              (id=100002)
OMVS.MUL02.ZFS                              (id=100001)
```

Figure 7-33 Displaying attached aggregates

The **`zfsadm lsaggr`** command lists all currently attached aggregates for zFS.

This command displays a separate line for each aggregate. Each line displays the following information:

- The aggregate name
- The aggregate ID number

You can use the **`zfsadm aggrinfo`** command to display information about the amount of disk space available on a specific aggregate or on all aggregates on a system.

7.34 List all file systems

```
ROGERS @ SC43:/>zfsadm lsfs -fast
OMVS.ALAN.ZFS
OMVS.TOLOD.ZFS
OMVS.YVES.ZFS
OMVS.CMP01.ZFS
OMVS.CMP01.ZFS
OMVS.CMP03.ZFS
OMVS.CMP04.ZFS
OMVS.MUL02.M02.ZFS.bak
OMVS.MUL02.m03.ZFS.bak
OMVS.MUL02.M04.ZFS.bak
OMVS.MUL02.M01.ZFS.bak
OMVS.MUL02.M01.ZFS
OMVS.MUL02.M04.ZFS
OMVS.MUL02.M02.ZFS
OMVS.MUL02.m03.ZFS
IOEZ00127I No file systems found for aggregate OMVS.MUL01.ZFS
```

Figure 7-34 Displaying zFS file systems

The **`zfsadm lsfs`** command lists all the file systems on a given aggregate or all attached aggregates.

If you specify an aggregate name that is used to retrieve file system information, the aggregate name is not case sensitive and it is always translated to upper case. If this option is not specified, the command displays information for all attached aggregates.

The **`-fast`** option causes the output of the command to be shortened to display only the aggregate name if it contains one or more file systems, or a message indicating that there are no file systems contained in the aggregate.

7.35 List all file systems (2)

```
$> zfsadm lsfs
IOEZ00127I No file systems found for aggregate  OMVS.MUL01.ZFS

IOEZ00129I Total of 1 file systems found for aggregate  OMVS.CMP04.ZFS
OMVS.CMP04.ZFS   RW (Mounted R/W)       9 K alloc      9 K quota On-line
Total file systems on-line 1; total off-line 0; total busy 0; total mounted 1

IOEZ00129I Total of 1 file systems found for aggregate  OMVS.CMP03.ZFS
OMVS.CMP03.ZFS   RW (Mounted R/W)       9 K alloc      9 K quota On-line
Total file systems on-line 2; total off-line 0; total busy 0; total mounted 2

IOEZ00129I Total of 1 file systems found for aggregate  OMVS.CMP02.ZFS
OMVS.CMP02.ZFS   RW (Mounted R/W)      58 K alloc     58 K quota On-line
Total file systems on-line 3; total off-line 0; total busy 0; total mounted 3

IOEZ00129I Total of 1 file systems found for aggregate  OMVS.CMP01.ZFS
OMVS.CMP01.ZFS   RW (Mounted R/W)       9 K alloc      9 K quota On-line
Total file systems on-line 4; total off-line 0; total busy 0; total mounted 4

IOEZ00129I Total of 4 file systems found for aggregate  OMVS.MUL02.ZFS
OMVS.m02c.ZFS RW (Mounted R/W)       9 K alloc      9 K quota On-line
OMVS.M02A.ZFS RW (Mounted R/W)      75 K alloc     75 K quota On-line
OMVS.M02B.ZFS RW (Mounted R/W)      76 K alloc     76 K quota On-line
OMVS.M02D.ZFS RW (Mounted R/W)       9 K alloc      9 K quota On-line
Total file systems on-line 8; total off-line 0; total busy 0; total mounted 8
```

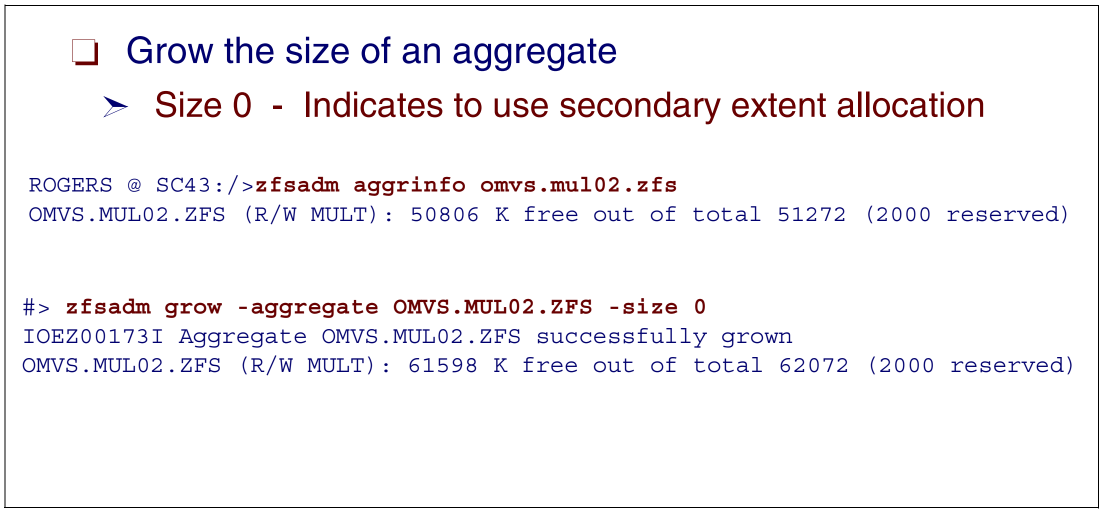
Figure 7-35 Listing zFS file systems with full description

The **zfsadm lsfs** command displays information about file systems in an aggregate. The file systems do not need to be mounted to use this command.

The **zfsadm lsfs** command displays the following information for a specified aggregate or all attached aggregates on a system:

- The total number of file systems contained in the aggregate.
- The file system's name (with a .bak extension, if appropriate).
- The type (RW for read-write, or BK for backup).
- If it is mounted or not.
- The allocation usage and the quota usage, in kilobytes.
- If the file system is online or not.
- The total numbers of file systems online, offline, busy, and mounted appear at the end of the output for all file systems.

7.36 Grow an aggregate



- Grow the size of an aggregate
  - Size 0 - Indicates to use secondary extent allocation

```
ROGERS @ SC43:/>zfsadm aggrinfo omvs.mul02.zfs
OMVS.MUL02.ZFS (R/W MULT): 50806 K free out of total 51272 (2000 reserved)


#> zfsadm grow -aggregate OMVS.MUL02.ZFS -size 0
IOEZ00173I Aggregate OMVS.MUL02.ZFS successfully grown
OMVS.MUL02.ZFS (R/W MULT): 61598 K free out of total 62072 (2000 reserved)
```

Figure 7-36 How to grow the size of an aggregate

The format utility formats the primary allocation and, if requested by using the *-size* parameter (with a value greater than the primary allocation), a single extension. An extension is a single call to the Media Manager to extend the data set to the size specified in the *-size* parameter. It is completely independent of the number of cylinders specified in the secondary allocation when the data set was defined (the secondary allocation could have been `0`).

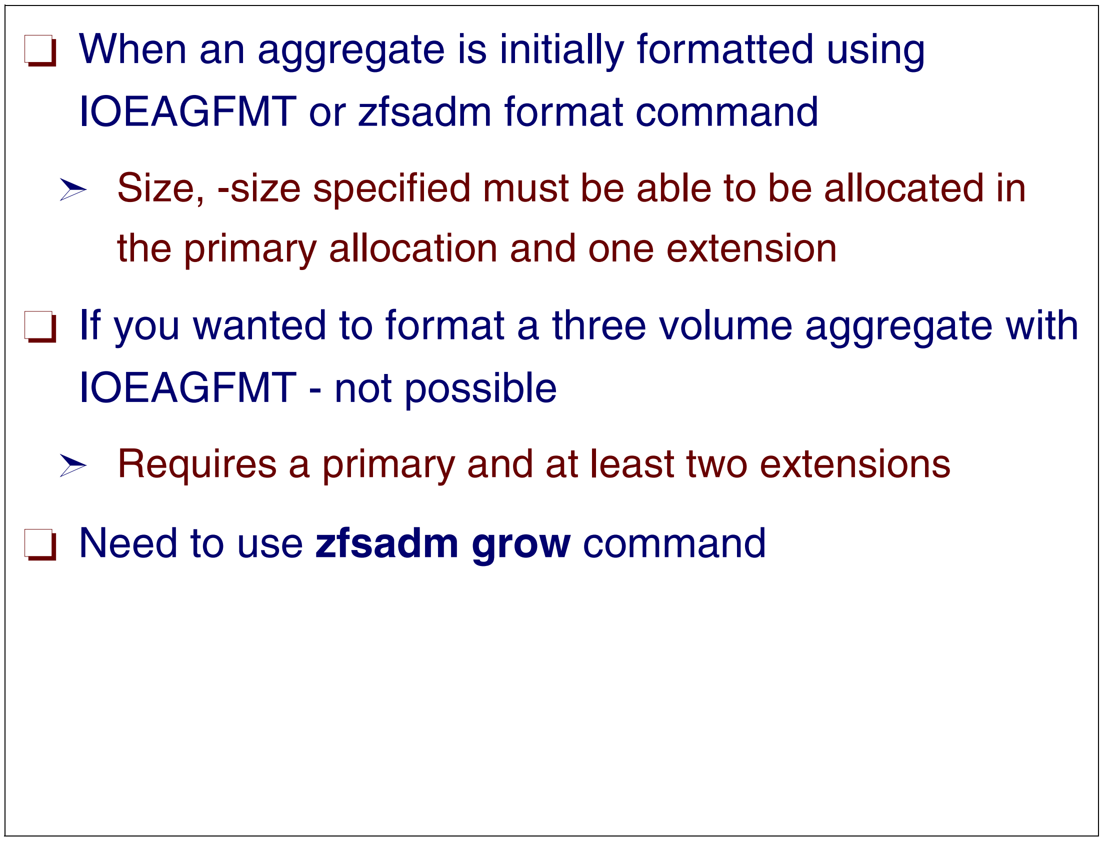
If a compatibility mode aggregate becomes full, the administrator can grow the aggregate (that is, cause an additional allocation to occur and format it to be part of the aggregate). This is accomplished with the **`zfsadm grow`** command. There must be space available on the volume(s) to extend the aggregate's VSAM Linear Data Set. The size specified on the **`zfsadm grow`** command must be larger than the current size of the aggregate.

For example, suppose a 2 cylinder (primary allocation, 3390) aggregate has a total of 179 8K blocks and a (potential) secondary allocation of 1 cylinder. 179 8K blocks is 1432K bytes. A **`zfsadm aggrinfo`** command for this aggregate might show 1296K with 136K reserved. This is a total of 1432K. A **`zfsadm grow`** command would need to specify a size greater than 1432 to actually grow the aggregate. **`zfsadm grow`** does this by calling DFSMS to allocate the additional DASD space. You may need to specify a few blocks larger than the current size before an allocation occurs because DFSMS may require some number of reserved blocks. For example, you may need to specify a size of 1441 before the extension actually occurs.

For a multi-file aggregate, if the sum of the quotas of all the file systems in an aggregate is greater than the physical space available in the aggregate, it is possible for a file system to run out of physical space before exceeding its quota. If this occurs, the application gets `ENOSPC` as a return code (the same return code it would get for exceeding its quota). The administrator can grow the aggregate (that is, cause an additional allocation to occur and format it to be part of the aggregate). This is accomplished with the **`zfsadm grow`** command. There must be space on the volume(s) to extend the aggregate's VSAM LDS. The size specified on the **`zfsadm grow`** command must be larger than the current size of the aggregate.

In the example in Figure 7-36, we used `-size 0`. Specifying a size of 0 indicates to use the secondary allocation.

7.37 New -grow option: z/OS V1R4



- When an aggregate is initially formatted using IOEAGFMT or zfsadm format command
  - Size, -size specified must be able to be allocated in the primary allocation and one extension
- If you wanted to format a three volume aggregate with IOEAGFMT - not possible
  - Requires a primary and at least two extensions
- Need to use **zfsadm grow** command

Figure 7-37 Using the -grow option when formatting aggregates

The VSAM linear data set must be formatted to be used as a zFS aggregate. There are two options available to format an aggregate:

- IOEAGFMT format utility
- **`zfsadm`** command

After you have allocated the space for an aggregate, the default size is the number of 8K blocks that fits into the primary allocation. You can specify a `-size` option giving the number of 8K blocks for the aggregate. If you specify a number that is less than (or equal to) the number of blocks that fits into the primary allocation, the primary allocation size is used. If you specify a number that is larger than the number of 8K blocks that fits into the primary allocation, the VSAM LDS is extended to the size specified. This occurs during its initial formatting.

When an aggregate is initially formatted using the IOEAGFMT format utility or the **`zfsadm format`** command, the formatting takes place as follows:

- The default size formatted is the number of blocks that will fit in the primary allocation.
- Using the `-size` parameter, if the number of blocks to be formatted is less than the default, it is rounded up to the default.
- If a number greater than the default is specified, a single extend of the VSAM LDS is attempted after the primary allocation is formatted.

7.38 New -grow option: z/OS V1R4 (2)

- Now, IOEAGFMT and **zfsadm format** provide the -grow option
  - Specifies the increment that will be used for extension when -size is larger than the primary allocation
  - Extends by -grow amount until -size is satisfied

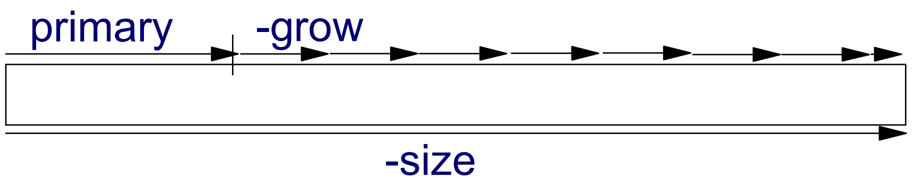


Figure 7-38 Specifying the -grow option

Since the `-size` parameter specified is only able to be allocated in the primary allocation and one extension, if you wanted to format a three volume aggregate with the IOEAGFMT utility, this was not possible because it requires a primary and at least two extensions. However, a new `-grow` option is provided with z/OS V1R4 for the IOEAGFMT utility and the **`zfsadm format`** command to allow specification of the increment that can be used for extension of the aggregate when `-size` is larger than the primary allocation. This allows the extension by the `-grow` amount until `-size` is satisfied.

-grow Specifies the number of 8K blocks that zFS uses as the increment for an extension when the `-size` option specifies a size greater than the primary allocation.

To illustrate the before and after using the `-grow` option, the following example has a VSAM LDS defined with 2 cylinders of primary space and 1 cylinder of secondary space:

```
//AYVIVAR2 JOB CLASS=J,MSGCLASS=A,NOTIFY=AYVIVAR
/*JOBPARM S=SC65
//P010 EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER(NAME(OMVS.TESTA.ZFS) VOLUMES(SBOX43) -
  LINEAR CYL(2,1) SHAREOPTIONS(2))
//
```

The file created has 2 extents: the first one corresponds to the primary space specified in the define process (30 tracks), and a second one with 30 tracks.

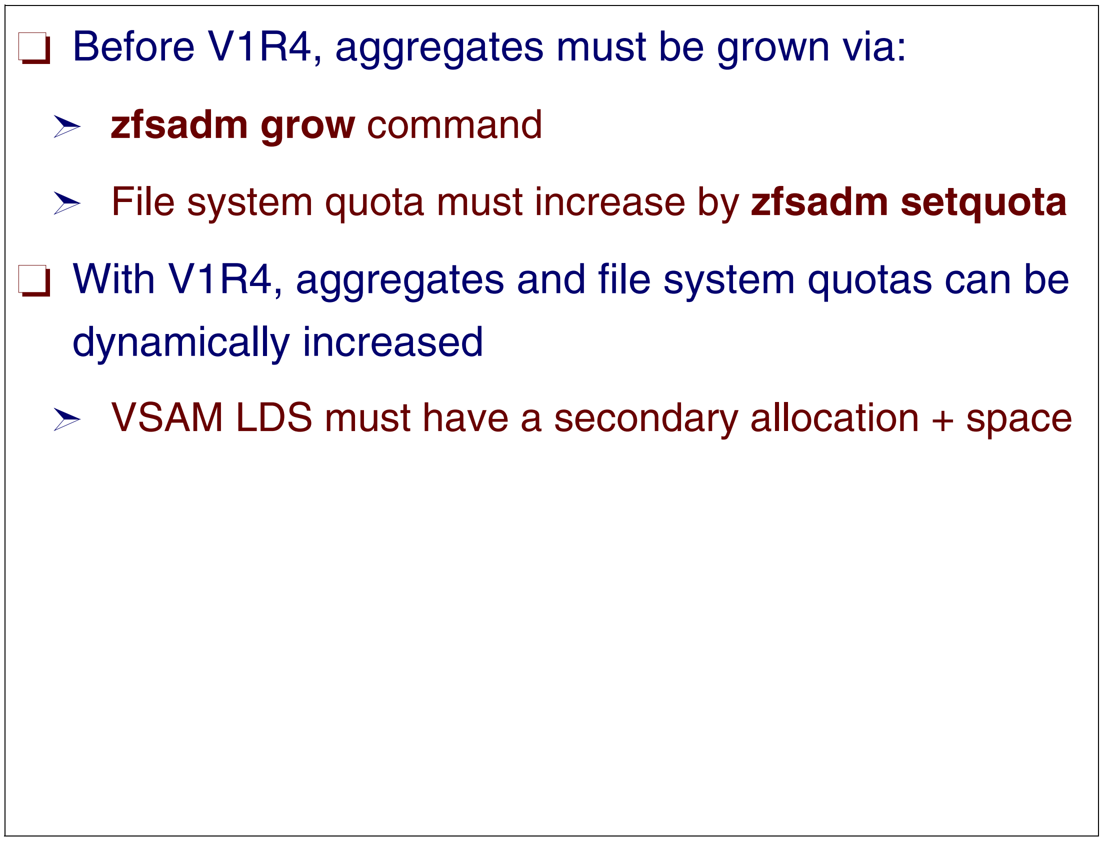
With the new -grow parameter, you are allowed to specify the increment that will be used for an extension size larger than the primary allocation. That is, after the primary space is allocated, multiple extensions of the amount specified by the -grow parameter rounded up to a multiple of the secondary space defined will be attempted until the total number of blocks specified by the -size parameter is satisfied.

Replacing the example shown previously, use the -grow parameter on the format process, as follows:

```
//AYVIVAR2 JOB CLASS=J,MSGCLASS=V,NOTIFY=AYVIVAR
/*JOBPARM S=SC65
//FORMAT EXEC PGM=IOEAGFMT,REGION=0M,
//       PARM=('-aggregate OMVS.TESTA.ZFS -size 276 -grow 90  -compat')
//SYSPRINT DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
```

Now the VSAM LDS has 3 extensions, the first corresponding to the primary space specified and the next two by the -grow amount.

7.39 Dynamic aggregate extension



- Before V1R4, aggregates must be grown via:
  - **zfsadm grow** command
  - File system quota must increase by **zfsadm setquota**
- With V1R4, aggregates and file system quotas can be dynamically increased
  - VSAM LDS must have a secondary allocation + space

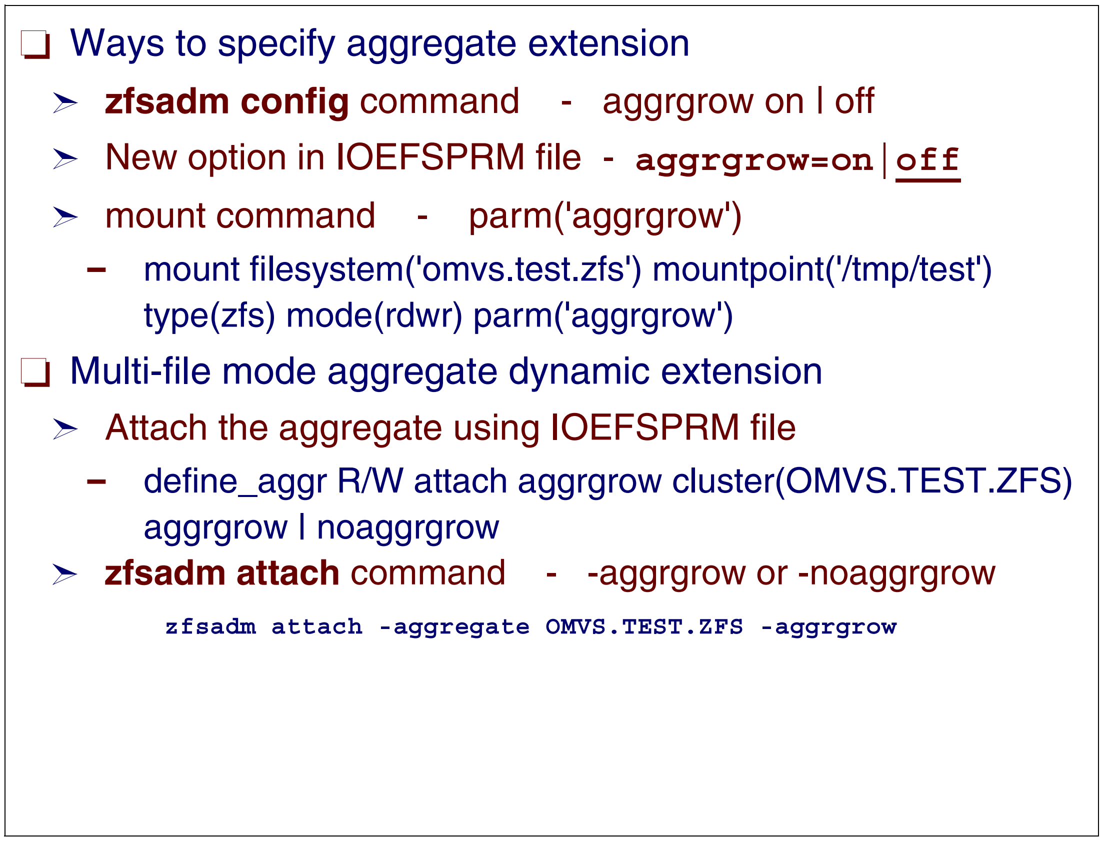
Figure 7-39 Growing the aggregate size dynamically

Before this change in z/OS V1R4, if aggregates became full they could only be grown by using the `zfsadm grow` command. You needed to specify a larger size or specify zero for the size to get a secondary allocation size extension.

z/OS V1R4 introduces the possibility of dynamically growing an aggregate if it becomes full. The aggregate is extended automatically when an operation cannot complete because the aggregate is full.

Important: To dynamically grow an aggregate when it becomes full, the VSAM LDS must have a secondary allocation and have space on the volume(s).

7.40 Dynamic aggregate extension aggrgrow


- Ways to specify aggregate extension
  - **zfsadm config** command - aggrgrow on | off
  - New option in IOEFSPRM file - **aggrgrow=on**|**off**
  - mount command - parm('aggrgrow')
    - mount filesystem('omvs.test.zfs') mountpoint('/tmp/test') type(zfs) mode(rdwr) parm('aggrgrow')
- Multi-file mode aggregate dynamic extension
  - Attach the aggregate using IOEFSPRM file
    - define_aggr R/W attach aggrgrow cluster(OMVS.TEST.ZFS) aggrgrow | noaggrgrow
  - **zfsadm attach** command - -aggrgrow or -noaggrgrow

```
zfsadm attach -aggregate OMVS.TEST.ZFS -aggrgrow
```

Figure 7-40 How to specify the dynamic aggregate grow option

An administrator can specify that an aggregate should be dynamically grown if it becomes full. This is specified by the `-aggrgrow` option on the **`zfsadm attach`** command, or the `aggrgrow` suboption of the `define_aggr` option of the IOEFSPRM file, or globally by the `aggrgrow` option of the IOEFSPRM file. The aggregate (that is, the VSAM Linear Data Set) must have secondary allocation specified when it is defined, and space must be available on the volume(s). The aggregate will be extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation will be redriven transparently to the application.

Dynamic aggregate extension can be enabled in the following ways:

- In the IOEFSPRM configuration file, you can dynamically extend an aggregate when it becomes full by specifying one of the following options:
  - Specify a new option `aggrgrow=on | off`. The default value is off.

    Note: The option specified here is the default if none of the following ways of specifying the `aggrgrow | noaggrgrow` options are used.

  - Specify either `aggrgrow | noaggrgrow` as a suboption on the `define_aggr` option for a multi-file system aggregate, as shown in the following definition:

```
define_aggr R/W attach aggrgrow cluster(OMVS.TEST.ZFS)
```

- Using the **`mount`** command, in the PARM keyword you can specify either `aggrgrow` or `noaggrgrow`, as shown in the following example:

  ```
  mount filesystem('omvs.test.zfs') mountpoint('/tmp/test') type(zfs) mode
  (rdwr) parm('aggrgrow')
  ```

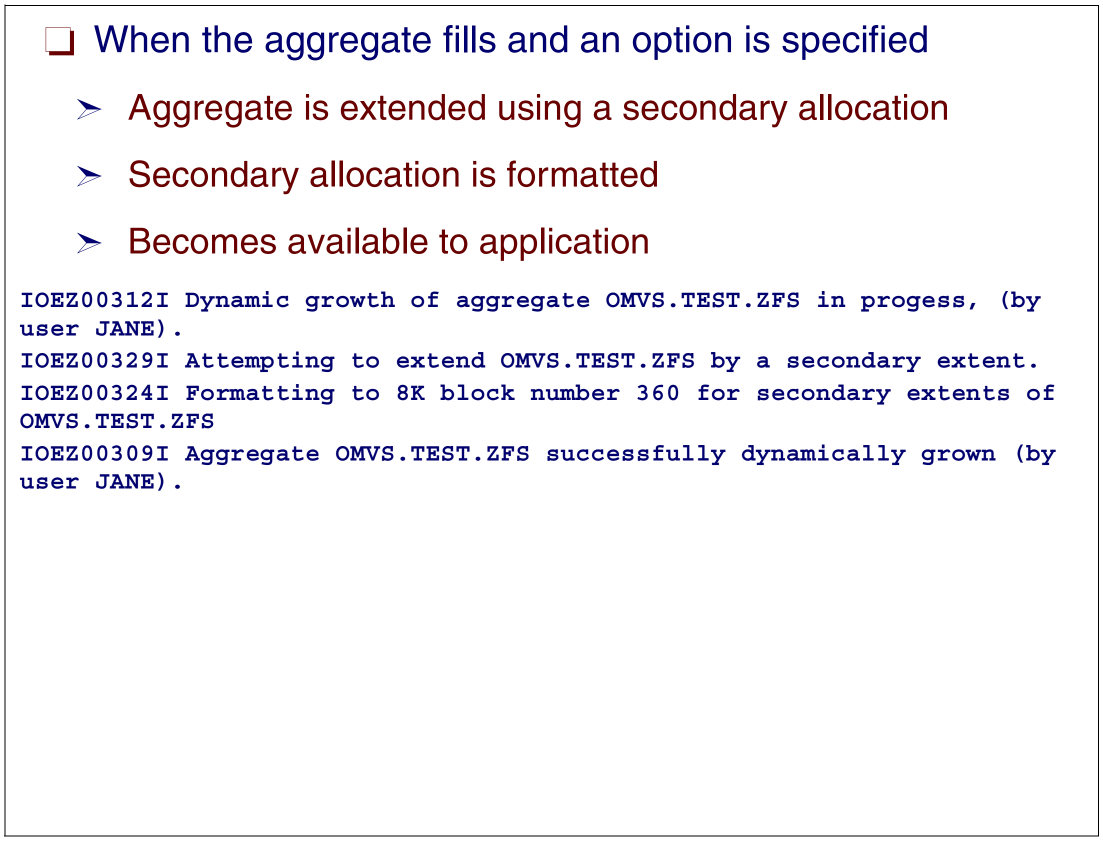
  Note: This `aggrgrow | noaggrgrow` option can only be used with compatibility mode aggregates.

- Using the **`zfsadm attach`** command, for attaching a multi-file system aggregate, you can specify either the `-aggrgrow` or `-noaggrgrow` option as shown in the following example:

  ```
  zfsadm attach -aggregate OMVS.TEST.ZFS -aggrgrow
  ```

- Using the **`zfsadm config`** command, you can dynamically change the configuration file option, `aggrgrow on | off`. This becomes the new default if no other option specification is in use.

7.41 Dynamic aggregate extension processing



❑ When the aggregate fills and an option is specified

- ➢ Aggregate is extended using a secondary allocation
- ➢ Secondary allocation is formatted
- ➢ Becomes available to application

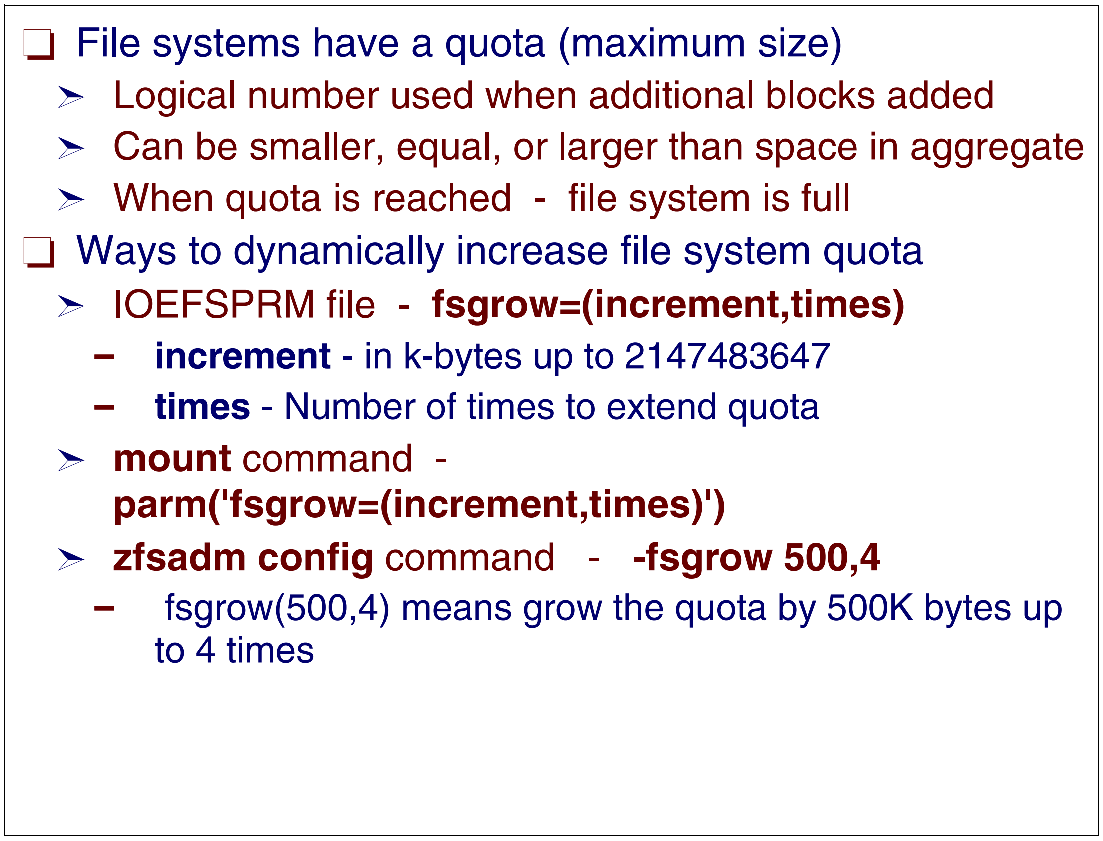
```
IOEZ00312I Dynamic growth of aggregate OMVS.TEST.ZFS in progess, (by
user JANE).
IOEZ00329I Attempting to extend OMVS.TEST.ZFS by a secondary extent.
IOEZ00324I Formatting to 8K block number 360 for secondary extents of
OMVS.TEST.ZFS
IOEZ00309I Aggregate OMVS.TEST.ZFS successfully dynamically grown (by
user JANE).
```

Figure 7-41 Example of growing an aggregate dynamically

When an aggregate fills and dynamic aggregate extension has been specified using one of the options, the aggregate is extended using secondary allocation extensions, and the extensions taken are formatted and become available transparently to the application. The messages issued indicating the process are the following:

```
IOEZ00312I Dynamic growth of aggregate OMVS.TEST.ZFS in progress, (by user AYVIVAR).
IOEZ00329I Attempting to extend OMVS.TEST.ZFS by a secondary extent.
IOEZ00324I Formatting to 8K block number 360 for secondary extents of OMVS.TEST.ZFS
IOEZ00309I Aggregate OMVS.TEST.ZFS successfully dynamically grown (by user AYVIVAR).
```

7.42 Dynamic file system quota increase



- ❑ File systems have a quota (maximum size)
  - ➢ Logical number used when additional blocks added
  - ➢ Can be smaller, equal, or larger than space in aggregate
  - ➢ When quota is reached - file system is full
- ❑ Ways to dynamically increase file system quota
  - ➢ IOEFSPRM file - **fsgrow=(increment,times)**
    - – **increment** - in k-bytes up to 2147483647
    - – **times** - Number of times to extend quota
  - ➢ **mount** command - **parm('fsgrow=(increment,times)')**
  - ➢ **zfsadm config** command - **-fsgrow 500,4**
    - – fsgrow(500,4) means grow the quota by 500K bytes up to 4 times

Figure 7-42 Specifying how to grow a zFS file system dynamically

The maximum size of a file system is known as its *quota*. This is a logical number that is compared against each time additional blocks are allocated to the file system. A quota can be smaller than, equal to, or larger than the space available in the aggregate. When the quota is reached, the file system indicates that it is full.

z/OS V1R4 introduces several ways to dynamically increase the file system quota if the file system becomes full. Dynamic file system quota increase can be specified in the following ways:

- A new option in the IOEFSPRM configuration file, `fsgrow=(increment,times)`, specifies whether file systems in a multi-file aggregate can have their quota dynamically extended. The value that is specified in this file becomes the default if no other way to specify this option is made. Where:

increment	The number of k-bytes to increase the quota. The maximum value that can be specified is 2147483647.
times	The number of times to extend the quota.

- Using the **`mount`** command, in the PARM keyword you can specify the file system quota extension of 500 KB for a maximum of four times, as `fsgrow(increment,times)`, as shown in the following example:

```
mount filesystem(zfstest) mountpoint('/tmp/zfstest') type(zfs) mode(rdwr)
parm('fsgrow(500,4)') noautomove
```

- Using the **`zfsadm config`** command, you can dynamically change the configuration file option by specifying, `-fsgrow increment times`, as shown in the following example:

  ```
  zfsadm config -fsgrow 500,4
  ```

  For example, `fsgrow(500,4)` means grow the quota by 500K bytes up to 4 times.

Displaying the quota

The following command shows the file system quota to be 1159, which was defined for a compatibility mode aggregate:

```
$> zfsadm lsquota -filesystem OMVS.TEST.ZFS
Filesys Name          Quota    Used  Percent Used  Aggregate
OMVS.TEST.ZFS         1159       9     0    11 = 146/1296 (zFS)
```

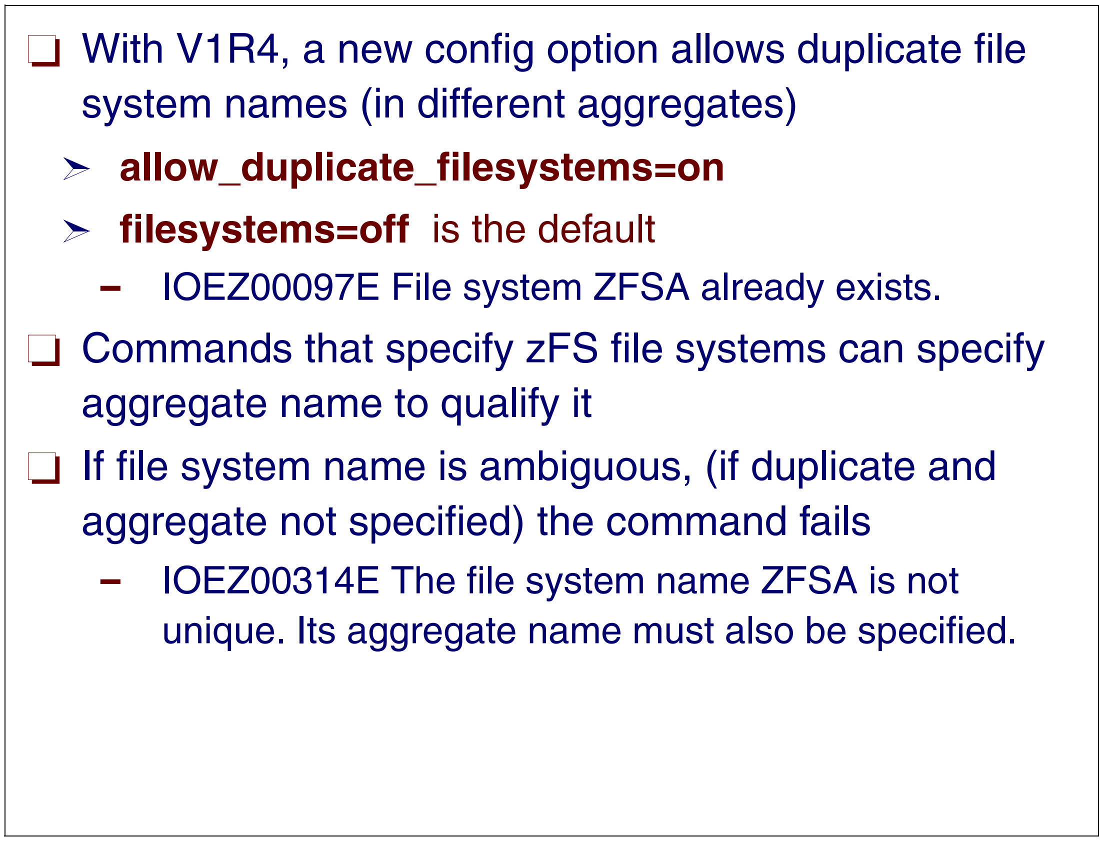
Compatibility mode aggregates

For a compatibility mode aggregate, only the `aggrgrow` specification is used to extend an aggregate size. The `fsgrow` option is ignored if it is specified. The quota increases by the size of the extension. For compatibility mode aggregates, the file system quota is dynamically increased based on a new aggregate size once it becomes full.

Multi-file mode aggregates

For a multi-file system aggregate, there may be multiple file systems in the aggregate. Therefore, if a file system becomes full, equal to its quota, an `fsgrow` option must be in place for the file system to then use the additional physical space that is available in the aggregate following the dynamic increase of the individual quota for the file system.

7.43 Duplicate file system names



- With V1R4, a new config option allows duplicate file system names (in different aggregates)
  - **allow_duplicate_filesystems=on**
  - **filesystems=off** is the default
    - IOEZ00097E File system ZFSA already exists.
- Commands that specify zFS file systems can specify aggregate name to qualify it
- If file system name is ambiguous, (if duplicate and aggregate not specified) the command fails
    - IOEZ00314E The file system name ZFSA is not unique. Its aggregate name must also be specified.

Figure 7-43 How to specify the use of duplicate file system names

Prior to z/OS V1R4, file system names were required to be unique among all the attached aggregates on a system. It is possible to create the same file system name on two different aggregates, but make sure to not have them attached at the same time. When the second aggregate is attached, an error message occurs and the duplicate file system is unavailable.

With z/OS V1R4, a new IOEFSPRM configuration file option allows duplicate file system names to be in different aggregates. The new option is:

```
allow_duplicate_filesystems=on
```

Commands that specify zFS file systems can now specify an aggregate name to qualify it, if more than one file system name exists. If a file system name is ambiguous, meaning the name is a duplicate and an aggregate name is not specified, the command fails.

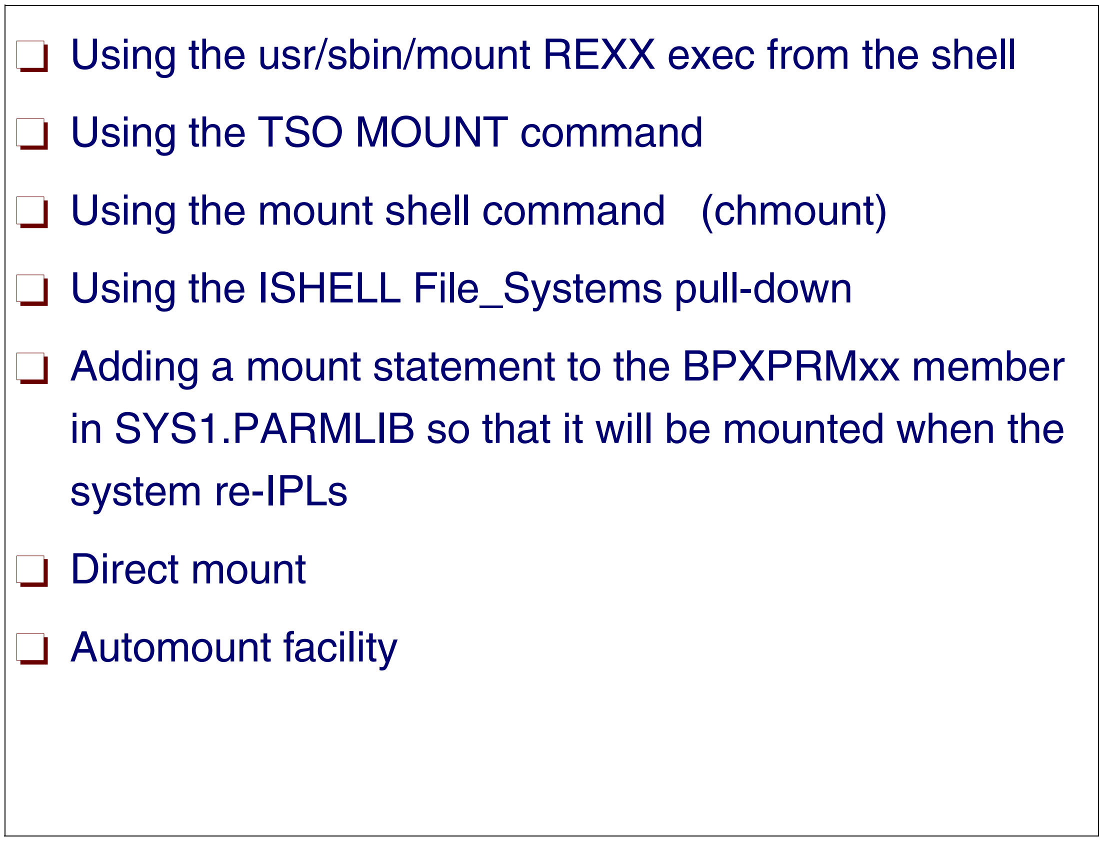
If you specify `allow_duplicate_filesystems=off`, which is the default, and then create a duplicate file system name, this request is denied and the following error message is issued:

```
IOEZ00097E File system ZFSA already exists.
```

It is possible to create the same file system name on two different aggregates by not having them attached at the same time. When the second aggregate is attached, the attach is successful but an error message occurs for the duplicate file system name, and it becomes unavailable, as follows:

```
IOEZ00314E The file system name ZFSA is not unique. Its aggregate name must also be
specified.
```

7.44 Mounting file systems



- Using the usr/sbin/mount REXX exec from the shell
- Using the TSO MOUNT command
- Using the mount shell command (chmount)
- Using the ISHELL File_Systems pull-down
- Adding a mount statement to the BPXPRMxx member in SYS1.PARMLIB so that it will be mounted when the system re-IPLs
- Direct mount
- Automount facility

Figure 7-44 How to mount zFS file systems

A file system is the entire set of directories and files, consisting of all HFS files shipped with the product and all those created by the system programmer and users. The system programmer (superuser) defines the root file system; subsequently, a superuser can mount other mountable file systems on directories within the file hierarchy. Altogether, the root file system and mountable file systems comprise the file hierarchy used by shell users and applications.

Figure 7-44 shows the different methods of mounting file systems.

The REXX exec **`/usr/sbin/unmount`** performs essentially the same functions that the TSO/E **`MOUNT`** command performs. You can run it from the shell.

Have an authorized user enter a TSO/E **`MOUNT`** command to logically mount the file system.

For hierarchical file systems, you can use the **`MOUNT`** command to logically mount, or add, a mountable file system to the file system hierarchy. This is the same command used by the REXX exec **`/usr/sbin/mount`**.

Using the File Systems pulldown from the ISHELL panel and Option 3, you can perform the mount for a file system.

The MOUNT statement defines the hierarchical file systems to be mounted at initialization and where in the file hierarchy they are to be mounted. It is up to the installation to ensure that all HFS data sets specified on MOUNT statements in the BPXPRMxx parmlib member are available at IPL time.

For a direct mount, you need to allocate an intermediate HFS data set 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 also need to add the HFS data set name and its mount point to the BPXPRMxx member of parmlib.)

Using the automount facility simplifies management of file systems. You do not need to mount most file systems at initialization and you do not need to request that operators perform mounts for other file systems. In addition, 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.

7.45 Mount and unmount

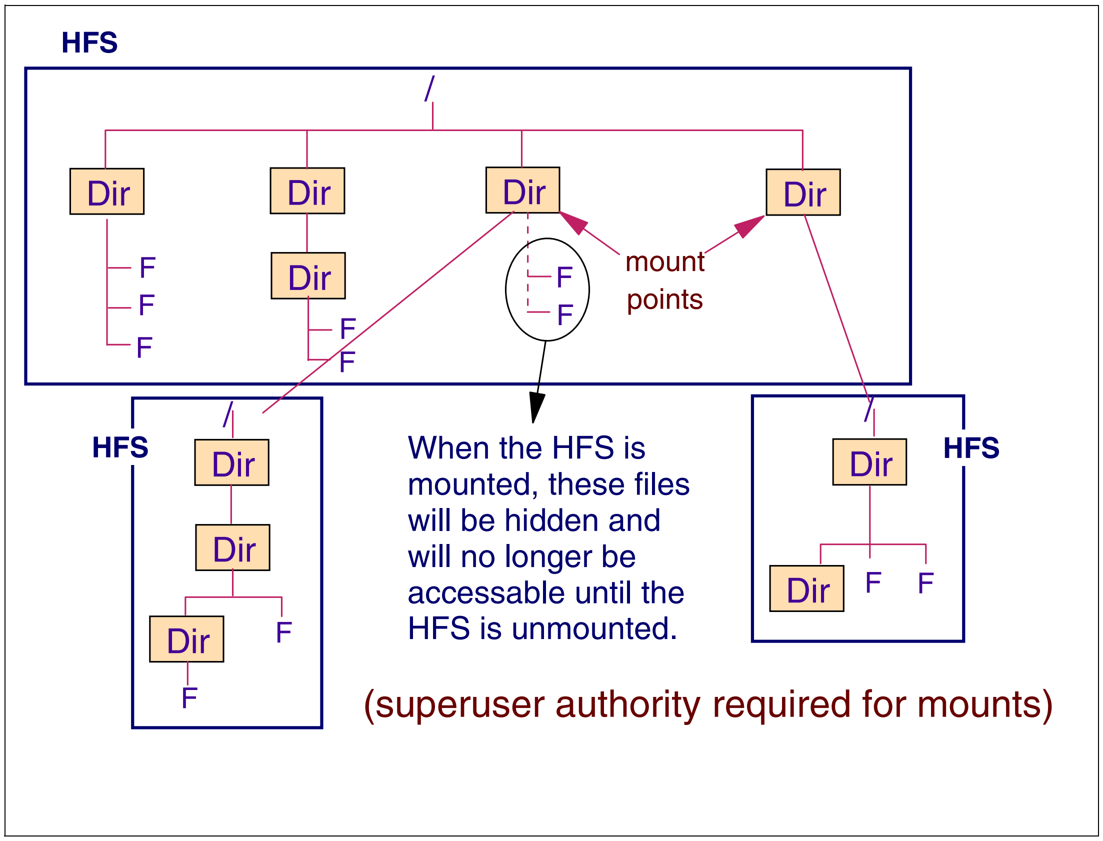

HFS
/
Dir
Dir
Dir
Dir
F
F
F
Dir
F
F
F
F
mount points
HFS
/
Dir
Dir
Dir
F
F
When the HFS is mounted, these files will be hidden and will no longer be accessable until the HFS is unmounted.
/
Dir
HFS
Dir
F
F
(superuser authority required for mounts)

Figure 7-45 Mounting of file system considerations

A file system must be mounted before it can be accessed. When z/OS UNIX is started, the root file system is mounted as the first file system. All other file systems included on the MOUNT statements in the BPXPRMxx members or mounted with the TSO command MOUNT will be mounted on the root file system or on other file systems.

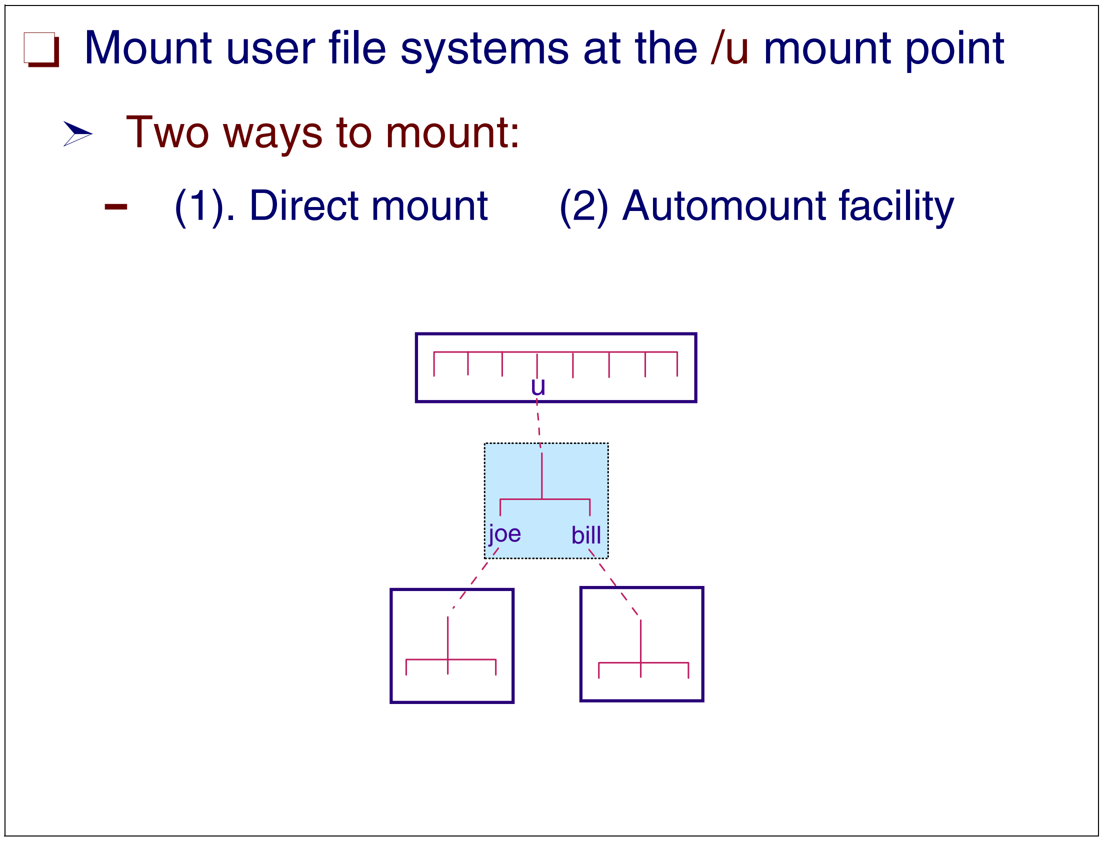
A file system can be mounted in read/write mode or in read-only mode. If it is mounted in read-only mode, nobody can update any files or directories in the file system. To change the mount mode, the file system must be unmounted and then remounted with a mode of read/write.

A file system must be mounted on a directory; this is called a *mount point.* Use empty directories as mount points. If a directory is not empty, the existing files will be overlapped by the file system which is mounted on the directory. When this file system is unmounted, the existing files will be accessible and visible again.

Superuser authority is required to mount or unmount a file system. That means the user must have a OMVS UID(0).

If the file system does not need to be updated, it could be mounted in a read-only mode. This will improve the performance. However, if a file system mounted as read-only needs updates, it must be unmounted and remounted again.

7.46 Managing user file systems



- Mount user file systems at the /u mount point
  - Two ways to mount:
    - (1). Direct mount (2) Automount facility

Figure 7-46 Managing the mounting of file systems of z/OS UNIX users

When a user requires an HFS or zFS file system to be accessed, you need to get it mounted at a mount point off of the root directory to make it available. The preferred place to mount all user HFS or zFS file systems is the **/u** mount point. In z/OS UNIX, there are two ways to accomplish this:

Direct mount Allocate an intermediate HFS data set 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 also need to add the HFS data set name and its mount point to the BPXPRMxx member of parmlib.

Automount You need to customize the automount facility to control all user file systems to automatically mount them when they are needed. This is the preferred method to manage user HFS data sets because it saves administration time.

The automount facility lets you designate directories as containing only mount points. This is the preferred method of managing user HFS file systems. As each of these mount points is accessed, an appropriate file system is mounted. The mount point directories are internally created as they are required. Later, when the file system is no longer in use, the mount point directories are deleted.

7.47 User file systems: Direct mount

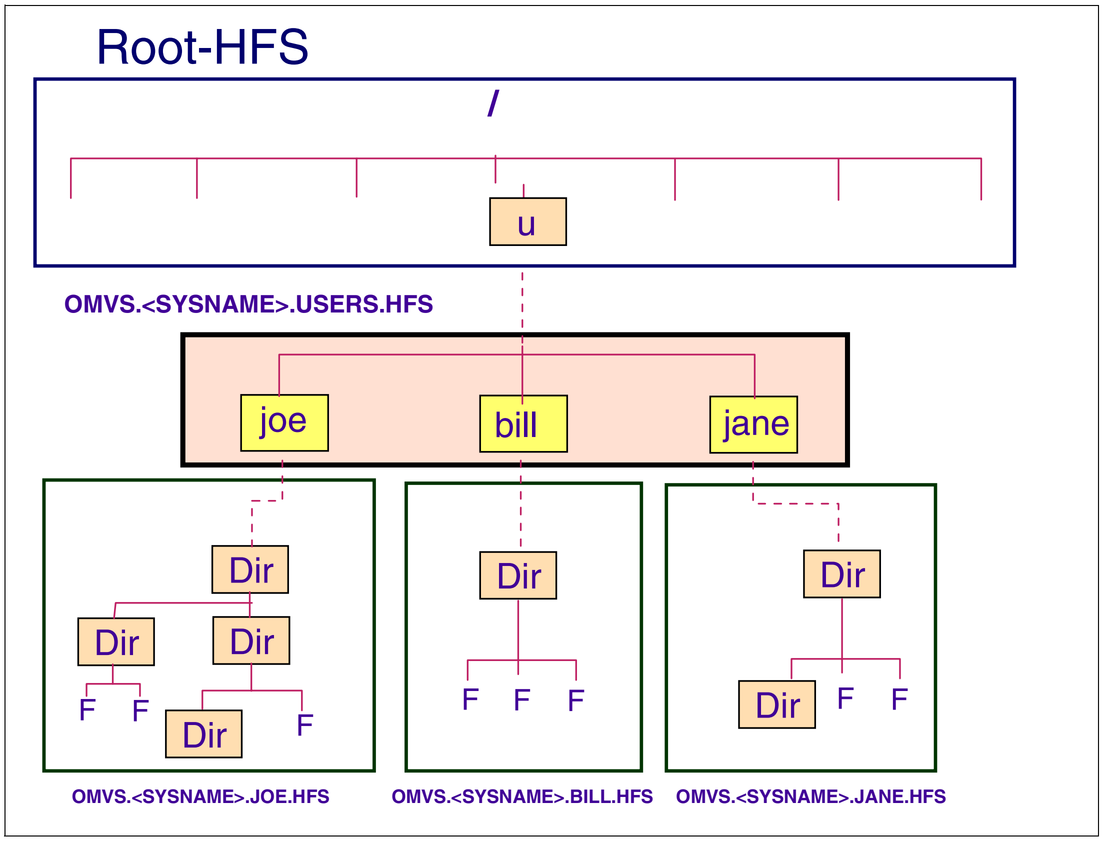


Figure 7-47 Mounting user file systems with direct mount

You should set up the root file system so that it does not require frequent changes. This can be accomplished by putting user data in user file systems. The **/u** directory contains the user home directories; if these directories are placed in separate file systems, most of the user data will be kept out of the Root file system.

The UNIX System Service programmer can choose to use only one HFS for all users, or use one HFS per user. If the user HFS is too small, you can mount some of the users to another user's HFS, or increase the user space for the HFS. This will make it easier to manage the HFS.

A recommendation is to name the user home directories /u/userid with the user ID in lowercase, for example /u/joe for user JOE as shown in the visual.

Following is a suggested method for creating user file systems:

- Leave the /u directory in the root file system empty.
- Create an HFS file called OMVS.<SYSNAME>.USERS.HFS.

  This file system will contain the home directories for all users, and will be mount points for the user file systems. The reason for keeping these home directories in a separate file system is to avoid updating the root file system each time a new home directory is created or deleted. The second qualifier of the DSNAME identifies the z/OS system image it relates to.

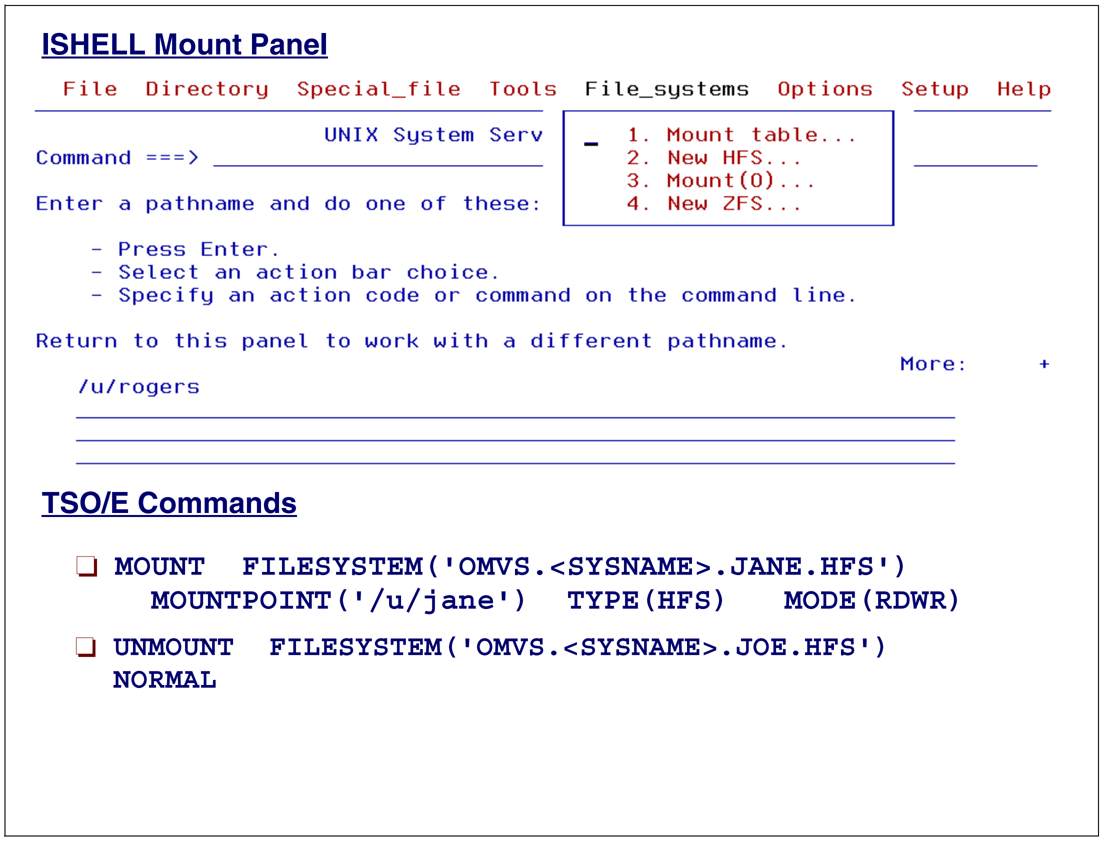
- Create a file system for each user or for a department, depending on what is best for the installation. The user file systems can be called OMVS.<SYSNAME>.<userid>.HFS.

  The user file systems will be mounted on the home directories in the OMVS.<SYSNAME>.USERS.HFS file system.

You should have a separate file system just to contain the user's home directories. These home directories will be empty, and they will act as mount points for the user file systems. Depending on how the installation is organized and on the need for user space, each user can have their own file system, or users in a department can share a system. It is easier to control space usage when each user has their own file system. It can be compared to users having their own z/OS data set for user data, or their own minidisk on VM. How large it should be depends entirely on who will use it and what the user will do. If multiple users share an HFS data set, it is not possible to guarantee space for each user. One user can dominate the space in a file system shared between multiple users.

When making plans for a file structure, it is important to think about a naming convention for the HFS data sets. If the automount facility will be used, one of the data set qualifiers should be equal to the directory name where the file system will be mounted.

7.48 Mounting file systems


ISHELL Mount Panel

```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                                        ┌─────────────────────┐
                          UNIX System Serv │ _  1. Mount table...  │
Command ===> ______________________________ │    2. New HFS...      │  __________
                                        │    3. Mount(O)...     │
Enter a pathname and do one of these:   │    4. New ZFS...      │
                                        └─────────────────────┘
    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                          More:     +
   /u/rogers
   ________________________________________________________________
   ________________________________________________________________
   ________________________________________________________________
```

TSO/E Commands

- ❑ `MOUNT FILESYSTEM('OMVS.<SYSNAME>.JANE.HFS') MOUNTPOINT('/u/jane') TYPE(HFS) MODE(RDWR)`
- ❑ `UNMOUNT FILESYSTEM('OMVS.<SYSNAME>.JOE.HFS') NORMAL`

Figure 7-48 How to mount file systems

The ISHELL is a powerful application based on ISPF. You can do many things with the menu shown at the top of Figure 7-48. The eight menu options have the following meanings:

File	Edit, browse, delete, copy, rename, print, run, and so on
Directory	List, create, rename, print, find string, and so on
Special_file	New FIFO, link, Attribute, delete, rename, and so on
Tools	Processes, Shell commands, run programs, and so on
File_systems	Mount, unmount, change attribute, allocate HFS, and so on
Options	Directory list, edit, browse, settings, and so on
Setup	User and Group Admin., create TTY, RACF permit Field, and so on
Help	Action code, Help, Keys help, About, and so on

A file system can be mounted by using the TSO/E **MOUNT** command or the ISHELL. Superuser authority is required for mounting or unmounting a file system.

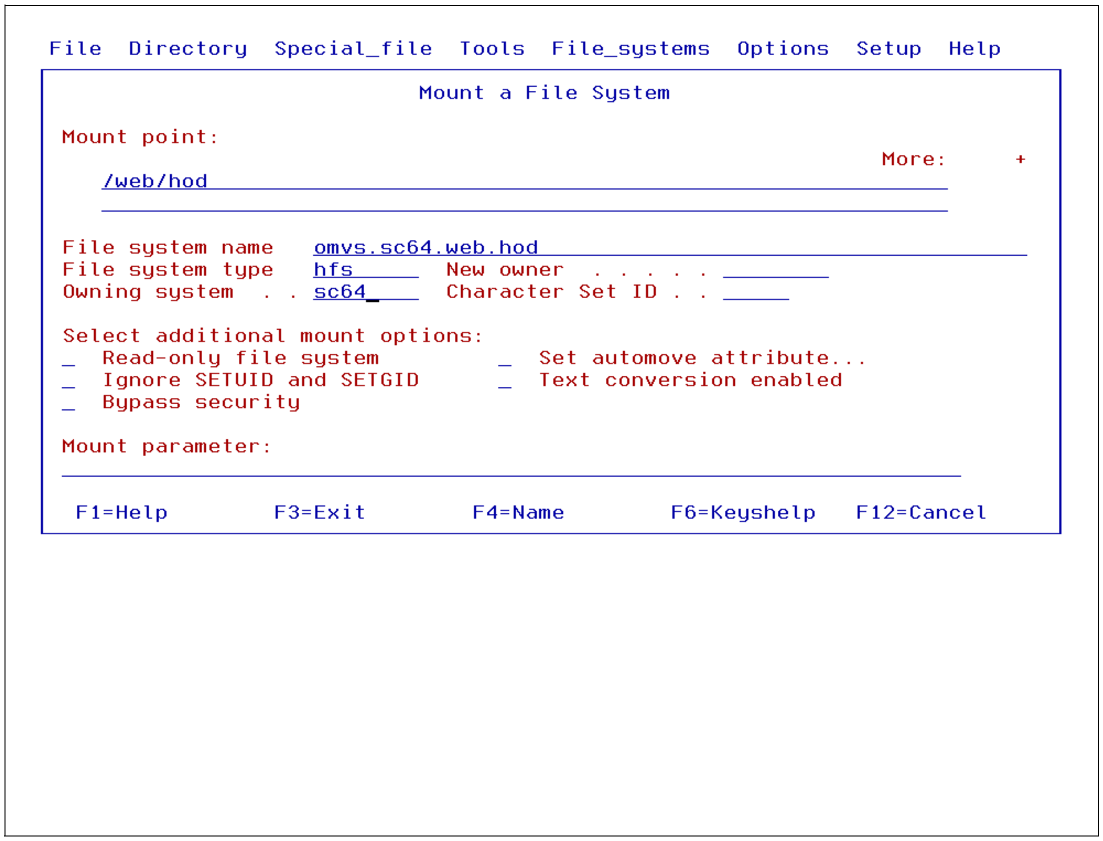
The options for the **MOUNT** command are the same as for the MOUNT statement in BPXPRMxx, except for an additional option called WAIT or NOWAIT. This option specifies whether to wait for an asynchronous mount to complete before returning.

The syntax for the **UNMOUNT** command is as follows:

```
UNMOUNT FILESYSTEM(file_system_name)
DRAIN | FORCE | IMMEDIATE | NORMAL | REMOUNT(RDWR | READ) | RESET
```

DRAIN	Specifies that an unmount drain request is to be made. The system will wait for all use of the file system to be ended normally before the unmount request is processed or until another UNMOUNT command is issued.
FORCE	Specifies that the system is to unmount the file system immediately. Any users accessing files in the specified file system receive failing return codes. All data changes to files in the specified file system are saved, if possible. If the data changes to the files cannot be saved, the unmount request continues and data is lost.
IMMEDIATE	Specifies that the system is to unmount the file system immediately. Any users accessing files in the specified file system receive failing return codes. All data changes to files in the specified file system are saved. If the data changes to files cannot be saved, the unmount request fails.
NORMAL	Specifies that if no user is accessing any of the files in the specified file system, the system processes the unmount request. Otherwise, the system rejects the unmount request. This is the default.
REMOUNT	(RDWR|READ) Specifies that the specified file system be remounted, changing its mount mode. REMOUNT takes an optional argument of RDWR or READ. If you specify either argument, the file system is remounted in that mode if it is not already in that mode. If you specify REMOUNT without any arguments, the mount mode is changed from RDWR to READ or READ to RDWR.
RESET	A reset request stops a previous UNMOUNT DRAIN request.
WAIT	Specifies that MOUNT is to wait for the mount to complete before returning. WAIT is the default.
NOWAIT	Specifies that if the file system cannot be mounted immediately (for example, a network mount must be done), then the command will return with a return code indicating that an asynchronous mount is in progress.

7.49 Option 3: Mount



```
 File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                         Mount a File System

 Mount point:
                                                               More:     +
    /web/hod


 File system name   omvs.sc64.web.hod
 File system type   hfs        New owner  . . . . . _________
 Owning system  . . sc64_____  Character Set ID . . ______

 Select additional mount options:
 _  Read-only file system         _  Set automove attribute...
 _  Ignore SETUID and SETGID      _  Text conversion enabled
 _  Bypass security

 Mount parameter:

  F1=Help        F3=Exit        F4=Name        F6=Keyshelp   F12=Cancel
```

Figure 7-49 Mounting file systems from the ISHELL with Option 3

The **ISHELL** command invokes the z/OS ISPF shell, a panel interface that helps you to set up and manage z/OS UNIX System Services functions such as mounting of file systems.

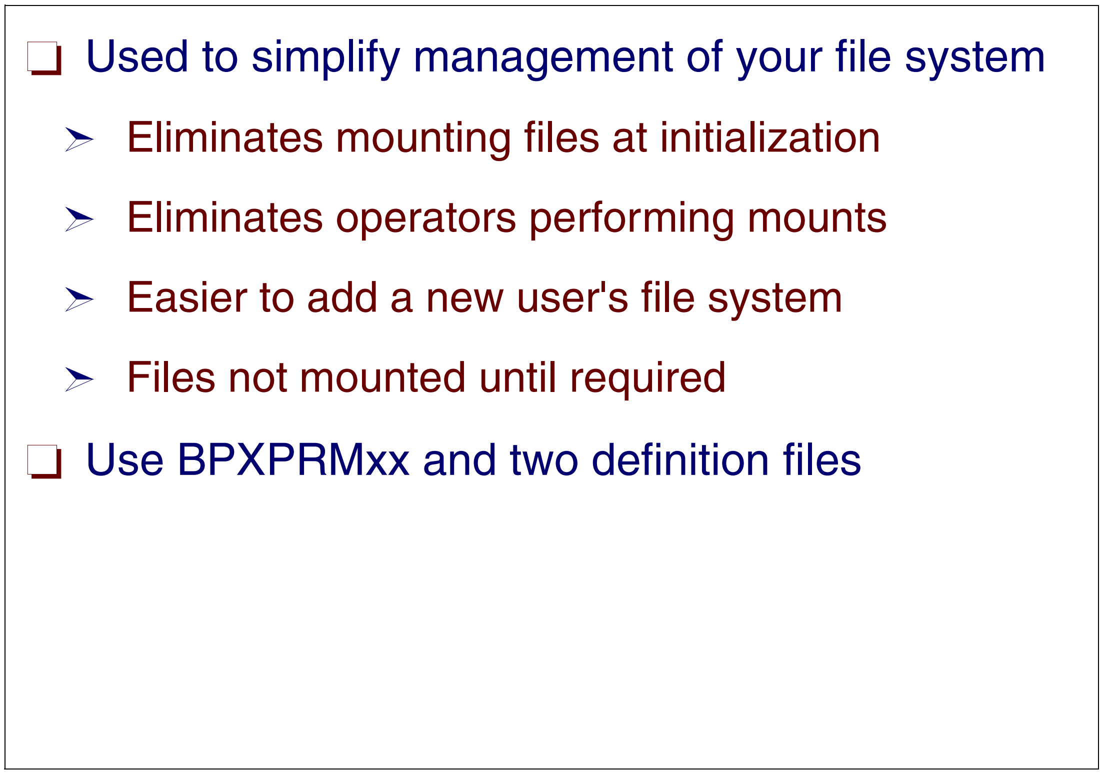
To mount a file system, you must be a superuser.

If you are a user with an MVS background, you may prefer to use the ISPF shell panel interface instead of shell commands or TSO/E commands to work with the file system. The ISPF shell also provides the administrator with a panel interface for setting up users for z/OS UNIX access, for setting up the root file system, and for mounting and unmounting a file system.

Select the options you require on the panel. 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.

Only one file system can be mounted at a directory (mount point) at any one time.

7.50 Automount facility



- ❑ Used to simplify management of your file system
  - ➢ Eliminates mounting files at initialization
  - ➢ Eliminates operators performing mounts
  - ➢ Easier to add a new user's file system
  - ➢ Files not mounted until required
- ❑ Use BPXPRMxx and two definition files

Figure 7-50 Using the automount facility

Use the automount facility to simplify management of your file system. With this facility, you do not need to mount most file systems at initialization and you do not need to request that operators perform mounts for other file systems. In addition, the facility simplifies the addition of new users because you can keep your parmlib specification stable. You can establish a simple automount policy to manage user home directories.

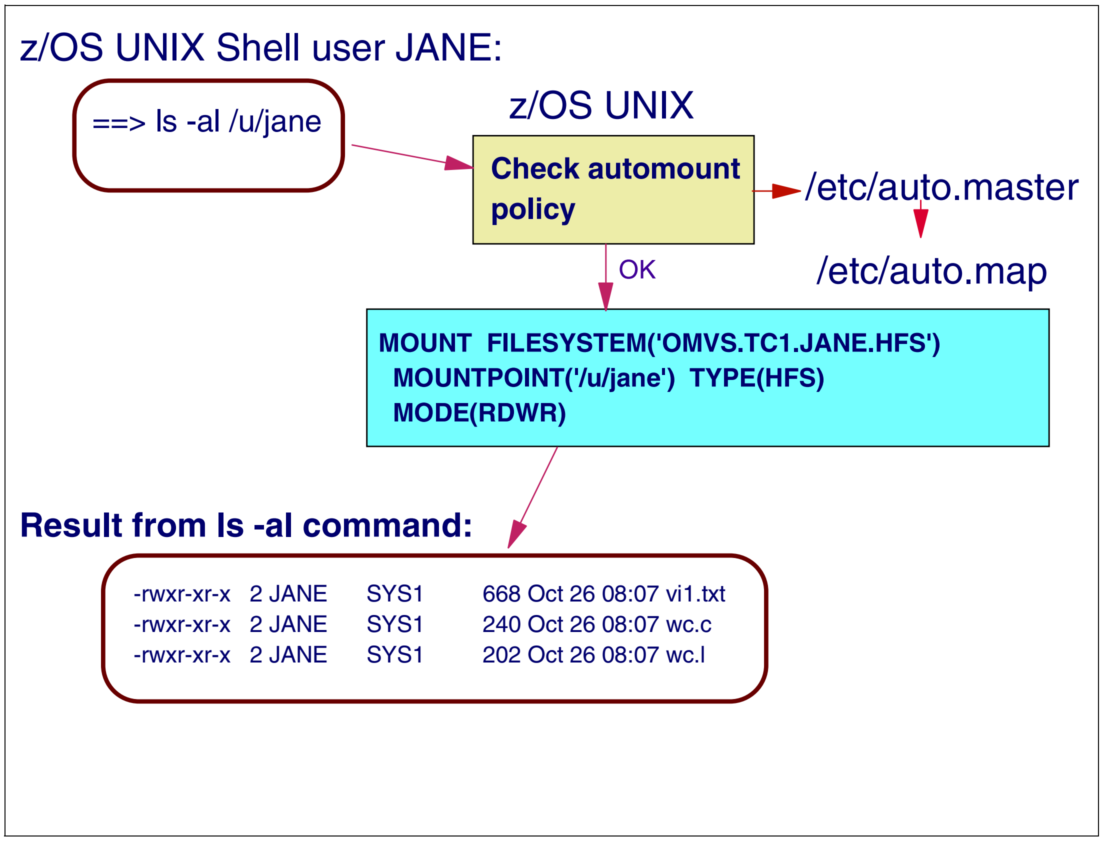
The automount facility also helps you to avoid consuming resources until they are requested. A file system that is managed by the automount facility remains unmounted until its mount point is accessed.

In addition, the automount facility helps you to reclaim system resources used by a mount if that file system has not been used for some period of time. You can specify how long the file system should remain mounted after its last use.

The automount facility lets you designate directories as containing only mount points. This is the preferred method of managing user HFS data sets. As each of these mount points is accessed, an appropriate file system is mounted. The mount point directories are internally created as they are required. Later, when the file system is no longer in use, the mount point directories are deleted.

Try to think of automount as an administrator that has total control over a directory. When a name is accessed in this directory, it looks up in its policy what file system is supposed to be associated with that name. If it finds one, it (logically) performs a **`mkdir`** followed by a **`mount`**, then quietly moves out of the way. Once out of the way, the root directory of that newly mounted file system is now accessed as that name.

7.51 Automount facility overview



```
MOUNT  FILESYSTEM('OMVS.TC1.JANE.HFS')
  MOUNTPOINT('/u/jane')  TYPE(HFS)
  MODE(RDWR)
```

```
-rwxr-xr-x   2 JANE     SYS1        668 Oct 26 08:07 vi1.txt
-rwxr-xr-x   2 JANE     SYS1        240 Oct 26 08:07 wc.c
-rwxr-xr-x   2 JANE     SYS1        202 Oct 26 08:07 wc.l
```

Figure 7-51 An overview of automount facility processing

The automount facility simplifies the management of mountable file systems.

With automount, an installation defines the directories that should be managed in a policy file. These directories must be the parent directories of the mount point directories. When a mount point directory (which is a subdirectory of a managed directory) is referenced in a pathname, the system will create the mount point directory and mount the proper file system.

The root directory cannot be managed by automount.

With automount, it is also possible to specify how long a file system should remain unreferenced before it is automatically unmounted. When a mounted file system is unmounted, the mount point directory will be deleted.

7.52 Automount setup

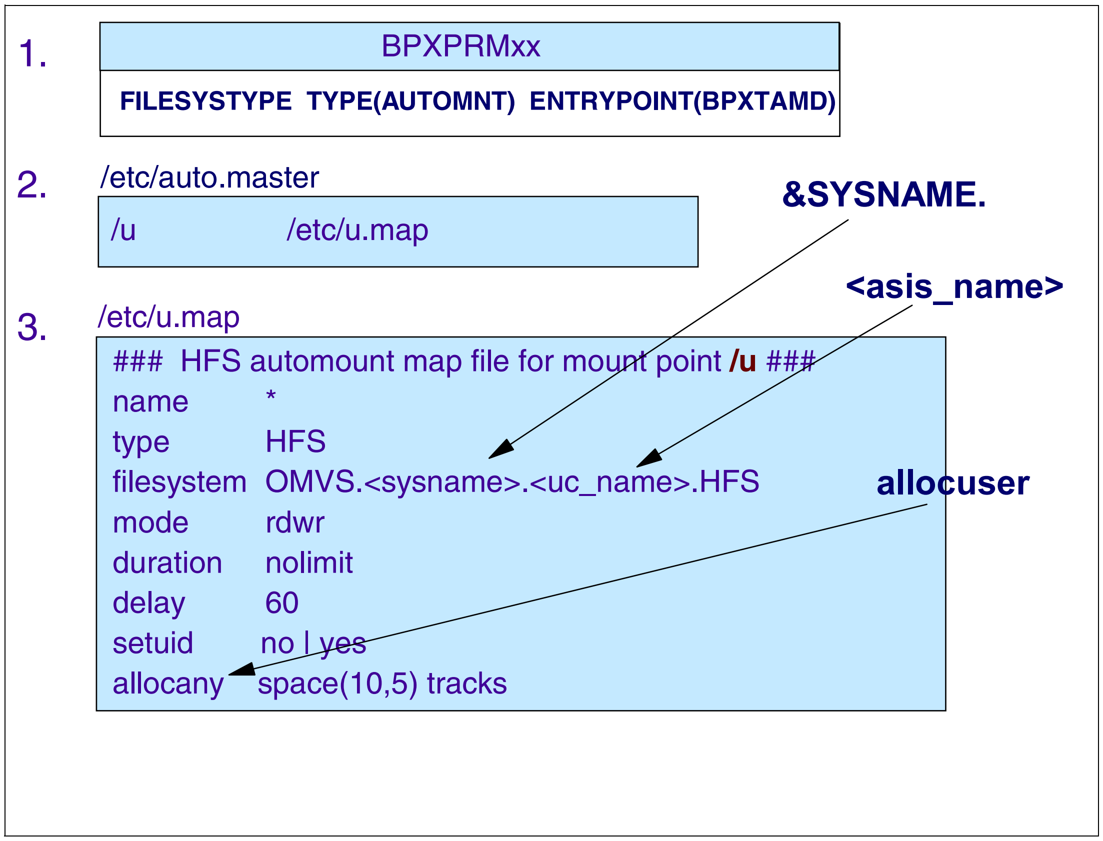

1.
BPXPRMxx
FILESYSTYPE TYPE(AUTOMNT) ENTRYPOINT(BPXTAMD)
2.
/etc/auto.master
/u /etc/u.map
3.
/etc/u.map
### HFS automount map file for mount point /u ###
name *
type HFS
filesystem OMVS.<sysname>.<uc_name>.HFS
mode rdwr
duration nolimit
delay 60
setuid no | yes
allocany space(10,5) tracks
&SYSNAME.
<asis_name>
allocuser

Figure 7-52 Automount facility policy setup

To use the automount facility, the following statement must be added to the BPXPRMxx member:

```
FILESYSTYPE TYPE(AUTOMNT) ENTRYPOINT(BPXTAMD)
```

Automount uses two HFS files for specifying the policy for which file systems should be automatically mounted when referenced:

- **/etc/auto.master** contains a list of directories to be managed, along with their MapName files.
- **/etc/u.map** is the MapName file for a directory.

  The MapName file contains the mapping between a subdirectory of a directory managed by automount and the mount parameters as follows:

  name The name of the mount point directory. An asterisk (*) specifies a generic entry for the automount-managed directory.

  type File system type. The default is HFS.

  file system Data set name of the file system to mount. Two special symbols are supported to provide name substitution:

  <asis_name> Used to represent the name exactly, as is.

  <uc_name> Used to represent the name in uppercase characters.

  These can be used when specifying a file system name or file system parameter that has a specific form, with the name inserted as a qualifier.

mode	Mount mode.
duration	The minimum amount of time in minutes to leave the file system mounted. The default is nolimit.
delay	The minimum amount of time in minutes to leave the file system mounted after the duration has expired and the file system is no longer in use. The default is 0.
setuid	Can be specified as yes or no. This will support or ignore the SETUID and SETGID mode bits on executable files loaded from the file system.

Note: The following attributes apply to a file system mounted with NOSETUID: SETUID and SETGID programs are not supported. That is, the UID or GID will not be changed when the program is executed.

The APF extended attribute is not honored.

The Program Control extended attribute is not honored.

New with z/OS V1R3

- **###** HFS automount map file for mount point /u ###. The use of the # symbol is new; it indicates a comment statement in the map file.
- **&SYSNAME** is new with z/OS V1R3. It indicates that system symbols are supported in the map file.
- **allocany** allocation-spec specifies the allocation parameters when using automount to allocate an HFS data set. `allocany` will cause an allocation if the HFS data set does not exist for any name looked up in the automount managed directory.
- **allocuser** allocation-spec specifies the allocation parameters when using automount to allocate an HFS data set. `allocuser` will cause an allocation to occur only if the name looked up matches the user ID of the current user.

Where:

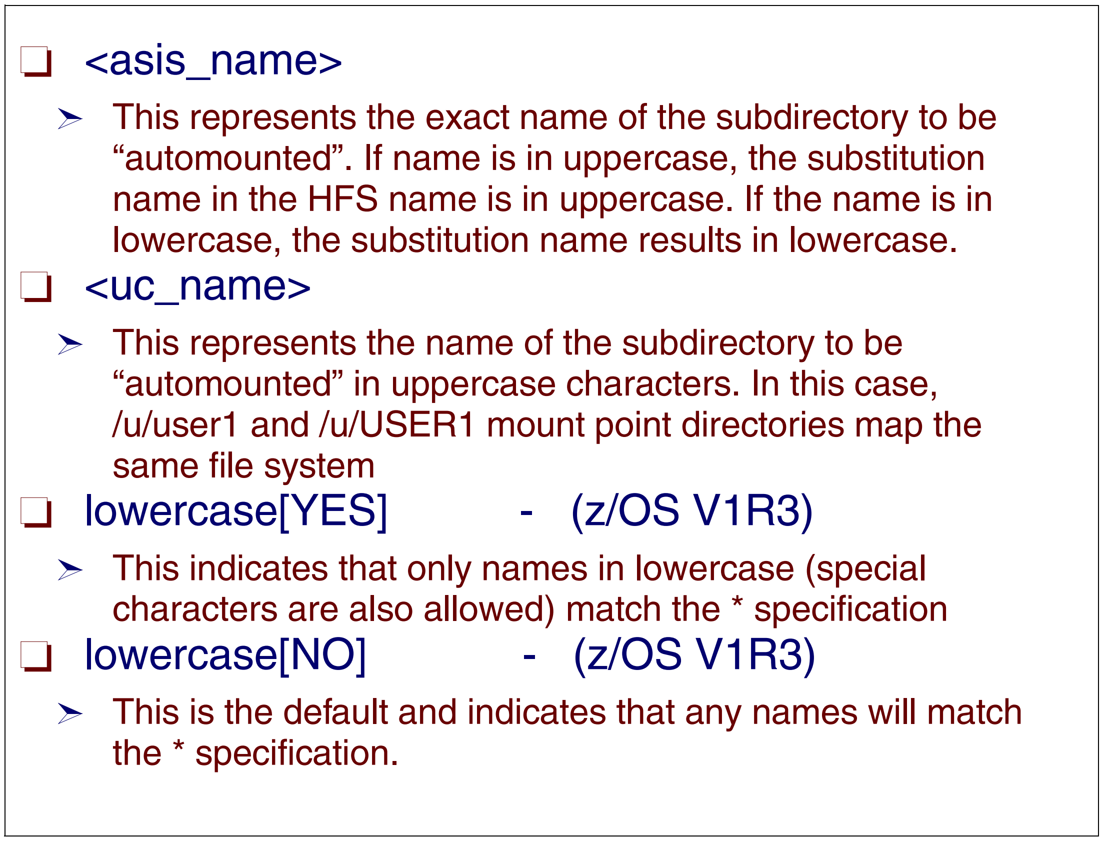
allocation-spec is a string that specifies allocation keywords. Keywords can be specified in the string, as follows:

```
space(primary-alloc[,secondary alloc])
cyl | tracks |  block(block size)
vol(volser[,volser]...)
maxvol(num-volumes)
unit(unit-name)
storclas(storage-class)
mgmtclas(management-class)
dataclas(data-class)
```

The next four keywords are automatically added:

```
dsn(filesystem)
dsntype(hfs)
dir(1)
new
```

7.53 Generic match on lower case names



- <asis_name>
  - This represents the exact name of the subdirectory to be "automounted". If name is in uppercase, the substitution name in the HFS name is in uppercase. If the name is in lowercase, the substitution name results in lowercase.
- <uc_name>
  - This represents the name of the subdirectory to be "automounted" in uppercase characters. In this case, /u/user1 and /u/USER1 mount point directories map the same file system
- lowercase[YES] - (z/OS V1R3)
  - This indicates that only names in lowercase (special characters are also allowed) match the * specification
- lowercase[NO] - (z/OS V1R3)
  - This is the default and indicates that any names will match the * specification.

Figure 7-53 Specification of the automount map file keywords

Four special symbols are supported to provide name substitution:

<asis_name>	Used to represent the name exactly, as is.
<uc_name>	Used to represent the name in uppercase characters.
<sysname>	Used to substitute the system name.
&SYSNAME.	Used to substitute the system name. This is new with z/OS V1R3.

Attention: IBM recommends that you use **&SYSNAME.**. <sysname> is only temporarily supported for compatibility.

You can use these when specifying a file system name or file system parameter that has a specific form with the name inserted as a qualifier.

lowercase [Yes|No] - This keyword is new with z/OS V1R3. It indicates the case for names that can match the * specification. This keyword is valid on any specification, but is only meaningful on the generic entry.

Yes	Only names composed of lowercase characters can match the * specification (numbers and special characters may also be used). When this is specified, uppercase characters are not allowed.
No	Any names can match the * specification. This is the default.

7.54 Activating automount

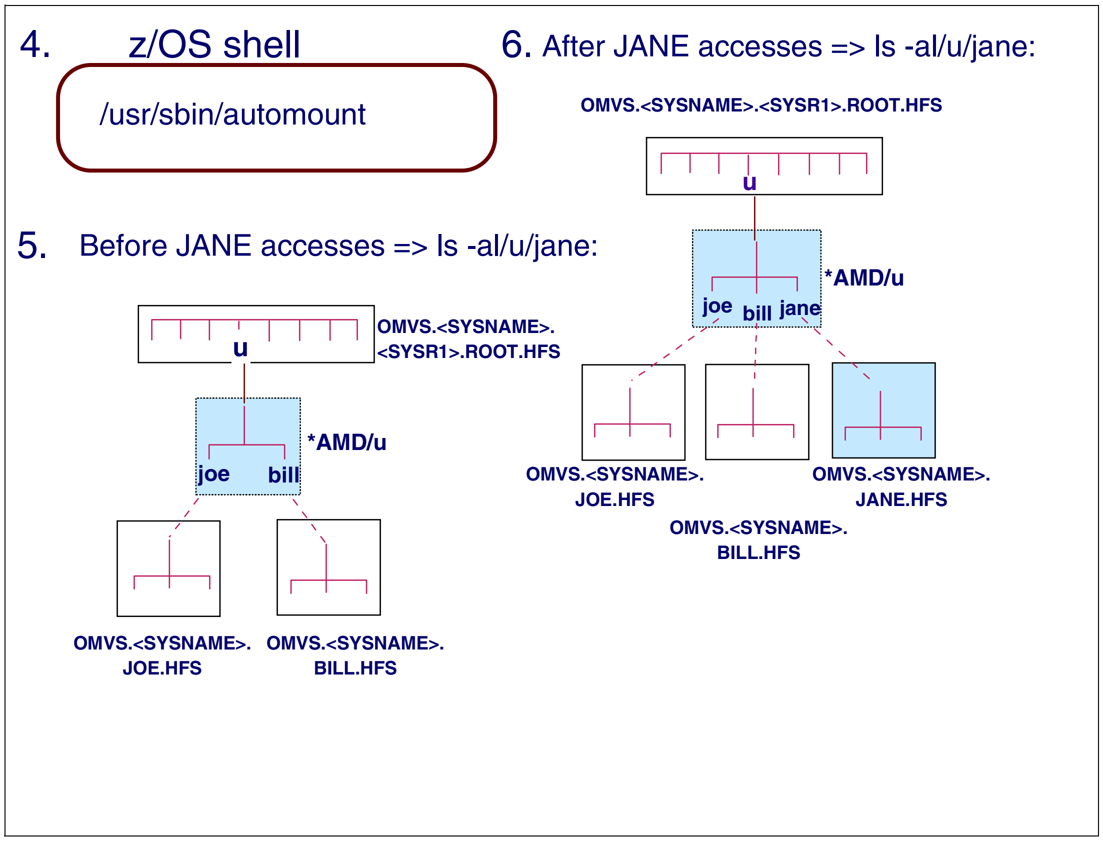


Figure 7-54 Activating the automount facility when a user accesses a file system

The **automount** command is used to configure the z/OS UNIX automount facility. When run with no arguments, automount reads the /etc/auto.master file to determine all directories that are to be configured for automounting and the filenames that contain their configuration specifications.

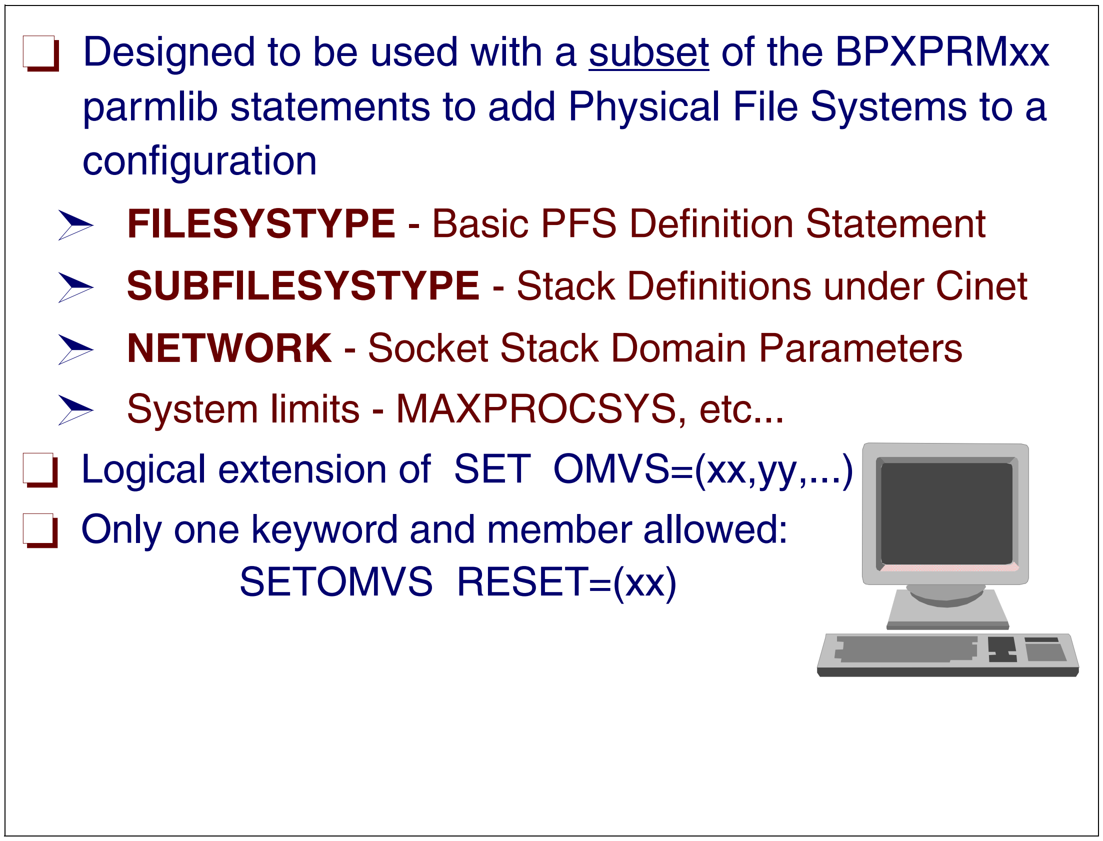
automount -s can be specified to check the syntax of the configuration file. No automount is performed when using the -s option.

The **automount** command requires superuser authority.The automount command is located in the /usr/sbin directory. The superuser should have this directory in their PATH environment variable.

When the HFS is first allocated for a new user and **automount** is used to dynamically allocate a mount point, the new mountpoint directory has permission bits of 700 and the owner is the superuser. A **chown** command will have to be issued to change the ownership.

After the **automount** command is done, the system will automatically mount a file system if the mount point directory is managed by the automount facility.

7.55 SETOMVS RESET=xx implementation



- Designed to be used with a subset of the BPXPRMxx parmlib statements to add Physical File Systems to a configuration
  - **FILESYSTYPE** - Basic PFS Definition Statement
  - **SUBFILESYSTYPE** - Stack Definitions under Cinet
  - **NETWORK** - Socket Stack Domain Parameters
  - System limits - MAXPROCSYS, etc...
- Logical extension of SET OMVS=(xx,yy,...)
- Only one keyword and member allowed:
  SETOMVS RESET=(xx)

Figure 7-55 Option to start a PFS with an operator command

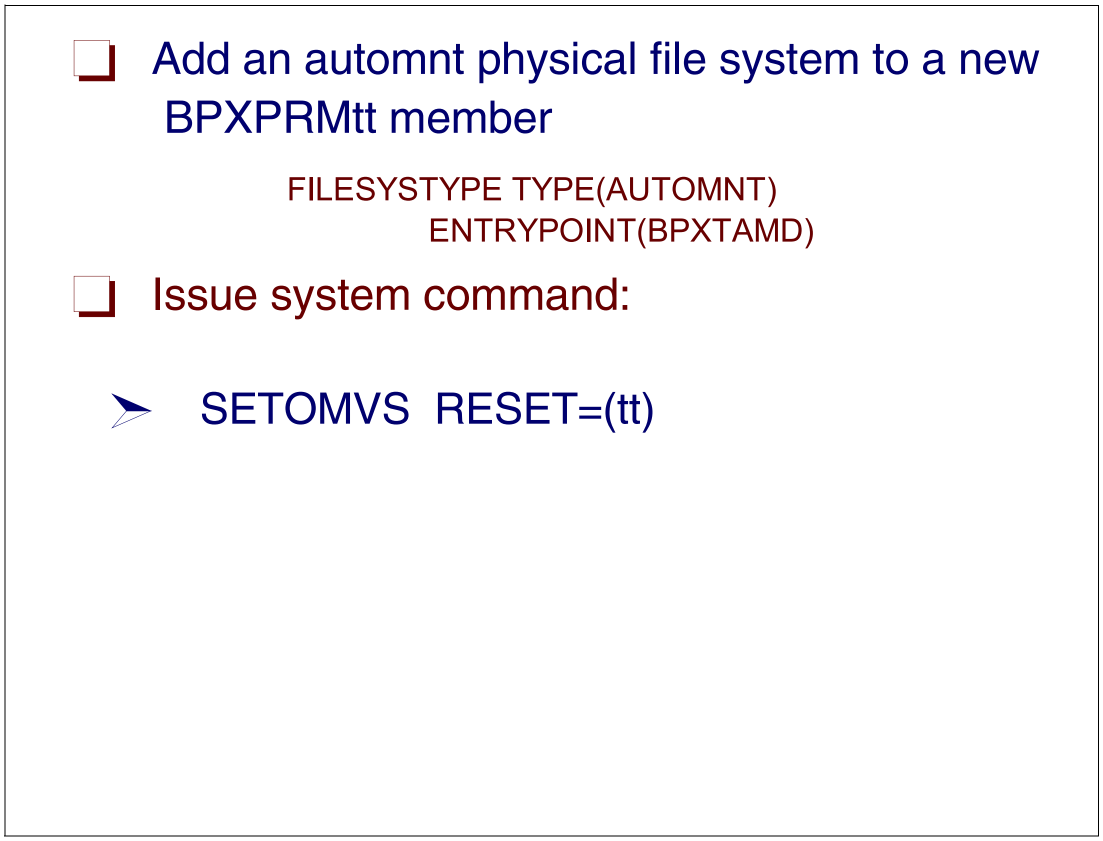
The **SETOMVS** command enables you to modify BPXPRMxx parmlib settings without re-IPLing. For example:

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

You can use the **SETOMVS RESET** command to dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements without having to re-IPL. However, if you change the values, a re-IPL will be necessary.

If you want to change the values, you will have to edit the BPXPRMxx member that is used for IPLs.

7.56 Issue SETOMVS command



- Add an automnt physical file system to a new BPXPRMtt member

  FILESYSTYPE TYPE(AUTOMNT)
  ENTRYPOINT(BPXTAMD)

- Issue system command:

  ➤ SETOMVS RESET=(tt)

Figure 7-56 Example of issuing the SETOMVS command

These steps show how to set up the automount facility to mount user file systems:

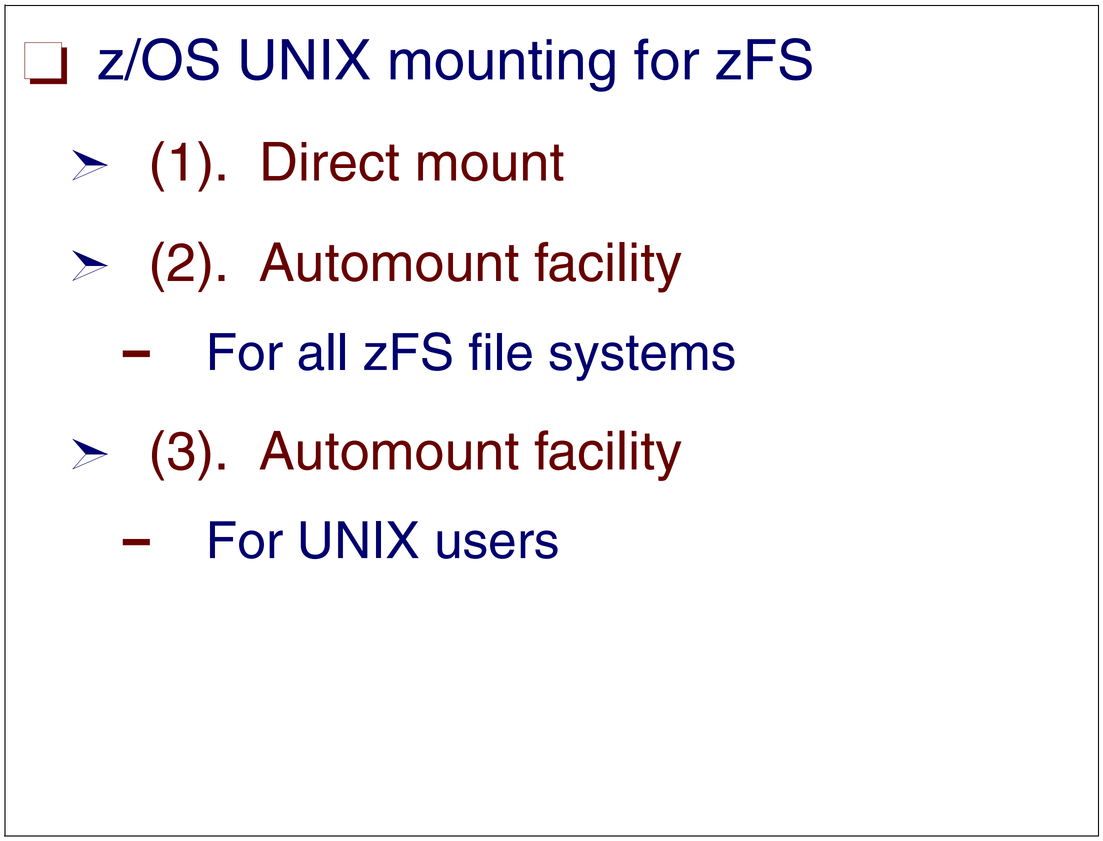
- Add the FILESYSTYPE statement to the BPXPRMxx member as follows:

```
FILESYSTYPE  TYPE(AUTOMNT)
      ENTRYPOINT(BPXTAMD)
```

- Use the **`SETOMVS RESET`** command to dynamically specify the new FILESYSTYPE statements; this changes the current system settings. (You cannot use the **`SETOMVS`** or **`SET OMVS`** command to specify the new statements.)
- Issue the **`SETOMVS RESET=`** command to dynamically create to AUTOMNT physical file system.

To make a permanent change, edit the BPXPRMxx member used for IPLs by placing the FILESYSTYPE statement into it.

7.57 How to mount zFS file systems



- z/OS UNIX mounting for zFS
  - (1). Direct mount
  - (2). Automount facility
    - For all zFS file systems
  - (3). Automount facility
    - For UNIX users

Figure 7-57 Methods to use when mounting zFS file systems

After you have created your zFS file systems, you need to get them mounted at a mount point off the root directory to make them available. The first consideration for mounting zFS file systems is to decide where in the root file system to create the starting mount point.

There are two types of aggregates that contain file systems to be mounted, as follows:

Compatibility mode	A zFS file system in a compatibility mode aggregate can be mounted, using the **`mount`** command, at an installation-created mount point. A zFS file system in a compatibility mode aggregate can also be AUTOMOVEed or automounted using the automount facility.
Multi-file mode	A multi-file system aggregate must be attached before a zFS file system can be created. After creating a directory for the mount point, you can use the **`mount`** command for the mount.

A preferred place to mount all user HFS data sets is a user directory under the /u user directory. Therefore, we describe both methods for the mounting of zFS file systems, since some installations may already be using direct mount while others may be using the automount facility. We suggest that you use one of the following methods:

1. Direct mount
2. Automount facility using zFS
3. Automount facility using zFS for users

7.58 Direct mount

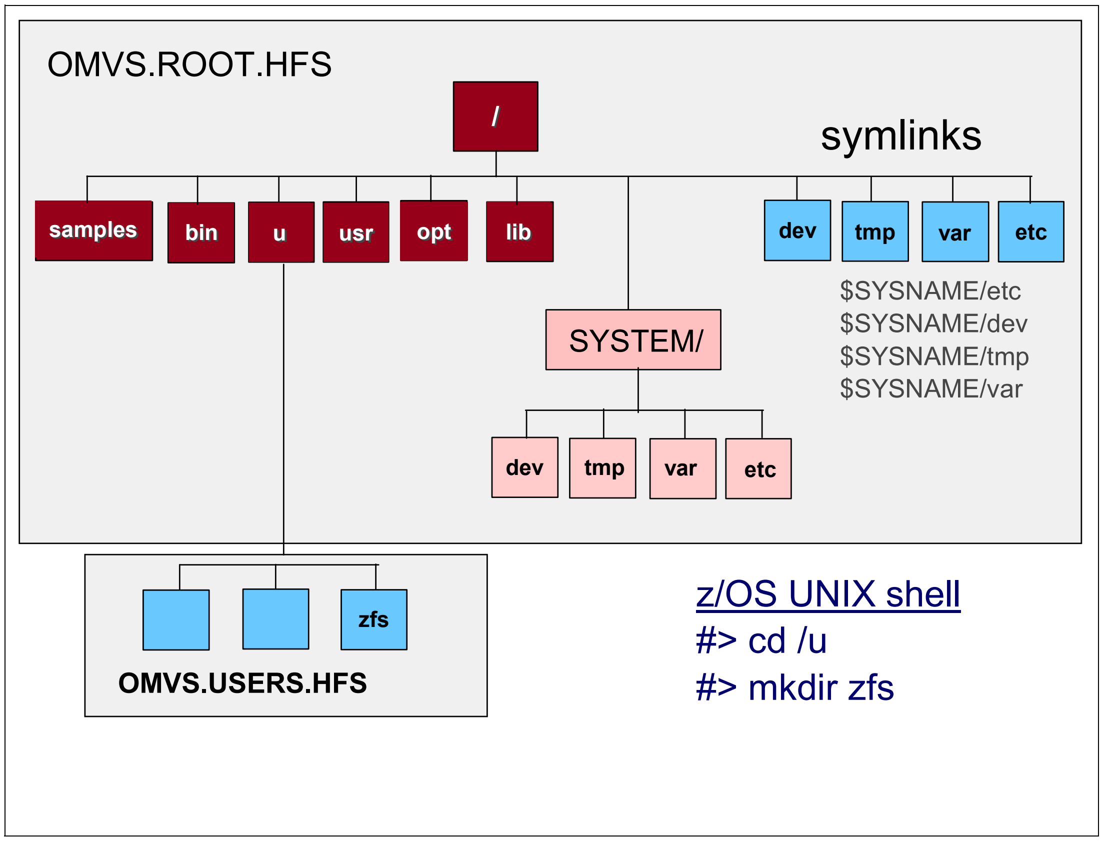


Figure 7-58 Using direct mount to mount zFS file systems

Using direct mount requires the allocation of an intermediate HFS or zFS data set, as shown in Figure 7-58. We named this HFS data set OMVS.USERS.HFS. It is to be mounted at the /u directory between the root file system and all user file systems.

Note: To make the mount of OMVS.USERS.HFS permanent every time an IPL occurs, you need to add the HFS data set name and its mount point to the BPXPRMxx member of parmlib.

Creating zFS mount points

Create a mount point in the OMVS.USERS.HFS data set by using the **mkdir** command as follows:

```
#> cd /u
#> mkdir zfs
```

You can now either add new directories for each zFS file system in OMVS.USERS.HFS, or add them off of the zfs directory mount point as shown in the figure.

7.59 Mounting zFS file systems

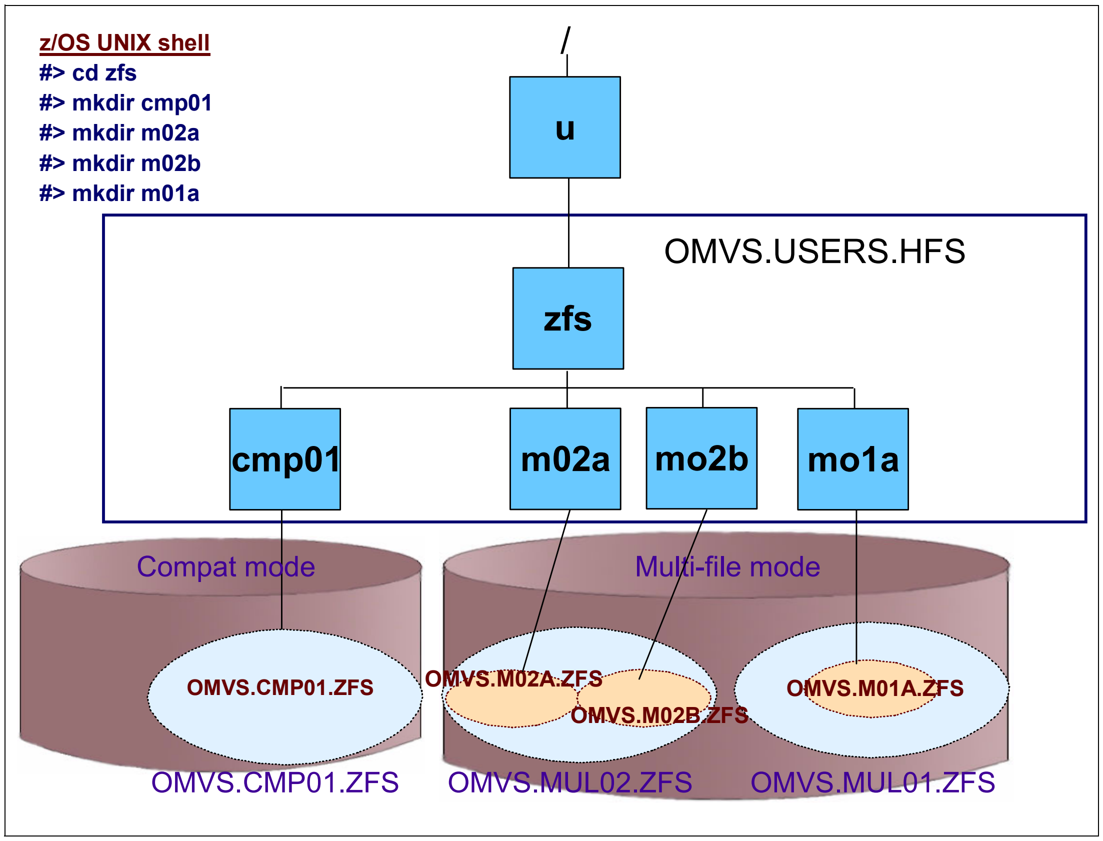


Figure 7-59 Mounting zFS file system with direct mount

In Figure 7-59 we show multiple zFS file systems mounted at specified mount points. For the multi-file system aggregates, we are using single-qualifier file system names.

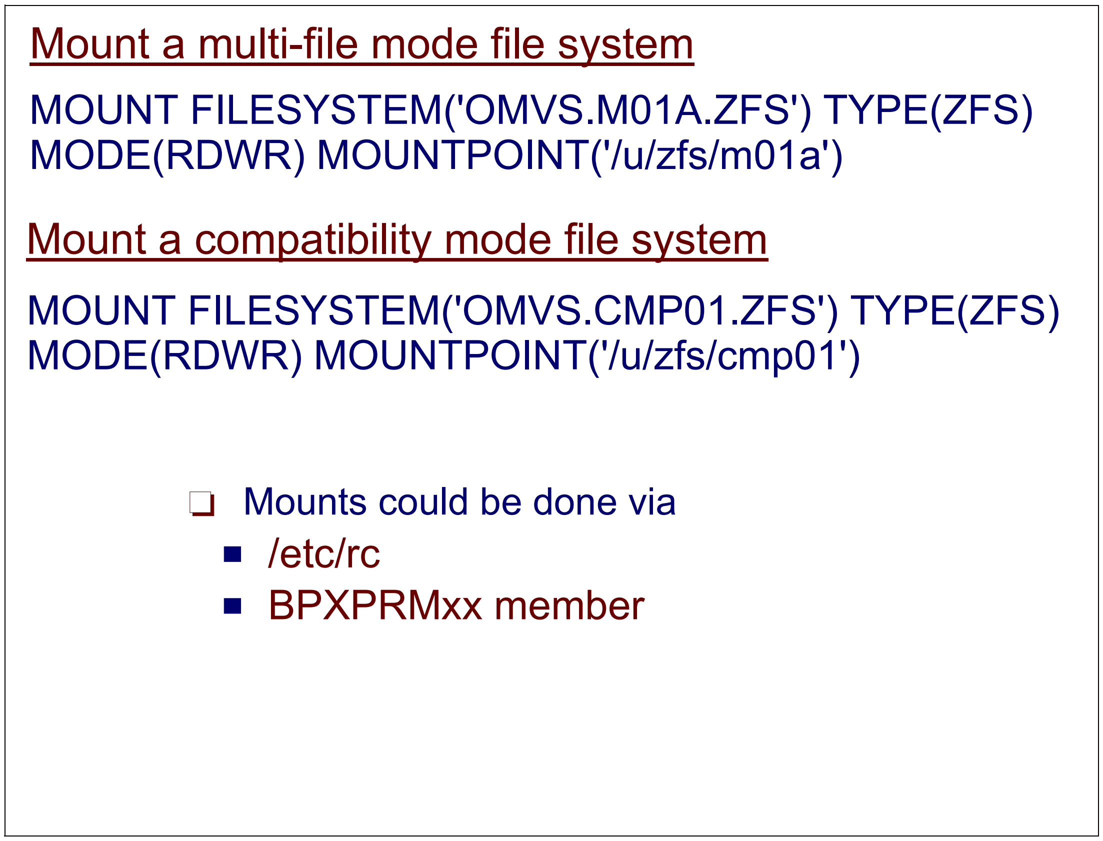
Attention: In this direct mount example, we are using a different naming convention for the aggregate names and file system names. We have also not shown the creation of the aggregates and file systems.

To create the directories and issue the mounts, the example continues as follows:

```
#> cd zfs
#> mkdir cmp01
#> mkdir m02a
#> mkdir m02b
#> mkdir m01a
```

As shown in the figure, there are two multiple file aggregates, one with one file system in it, OMVS.M01A.ZFS, and the other with two file systems, OMVS.M02A.ZFS and OMVS.M02B.ZFS. The compatibility mode aggregate, of course, has only one file system, which is named the same as the aggregate name.

7.60 MOUNT command from TSO/E



Mount a multi-file mode file system

MOUNT FILESYSTEM('OMVS.M01A.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/m01a')

Mount a compatibility mode file system

MOUNT FILESYSTEM('OMVS.CMP01.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/cmp01')

- Mounts could be done via
  - /etc/rc
  - BPXPRMxx member

Figure 7-60 Issuing the mount command from TSO/E

Once your zFS file systems have been defined, you can make the following decisions on how to mount them:

- You can issue the MOUNT command from TSO/E for the compatibility mode file system as follows:

```
MOUNT FILESYSTEM(‘OMVS.CMP01.ZFS’) TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/cmp01')
```

- Issue the MOUNT command from TSO/E for the multi-file file system as follows:

```
MOUNT  FILESYSTEM(' OMVS.M02A.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/m02a')
MOUNT  FILESYSTEM(' OMVS.M02B.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/m02b')
MOUNT  FILESYSTEM(' OMVS.M01A.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/m01a')
```

Note: zFS multi-file file system names can be any number of qualifiers.

- IPL the system and use the startup of z/OS UNIX to have them mounted.
- Use the MOUNT command from TSO/E or the **`mount`** command from the OMVS shell.

Note: There are some further possibilities for mounting file systems. You can use option 3 in the File_systems action menu of the ISHELL, for example.

7.61 Automount policy using /z

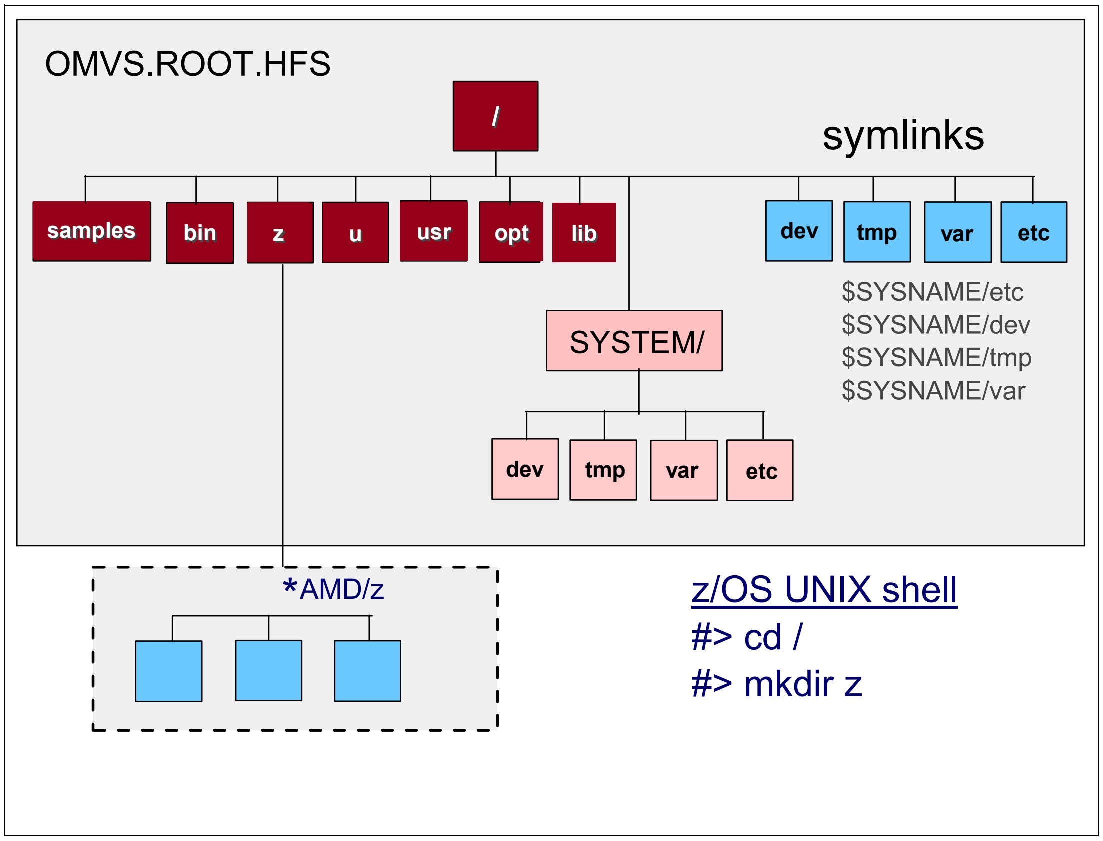


Figure 7-61 Using the automount policy for direct mount of zFS file systems

If your installation already has an existing automount policy using /u, you can consider the implementation we used for the mounting of zFS file systems into the z/OS UNIX hierarchy.

The first step is to create a new directory in the root. In our example, this is directory /z shown in Figure 7-61.

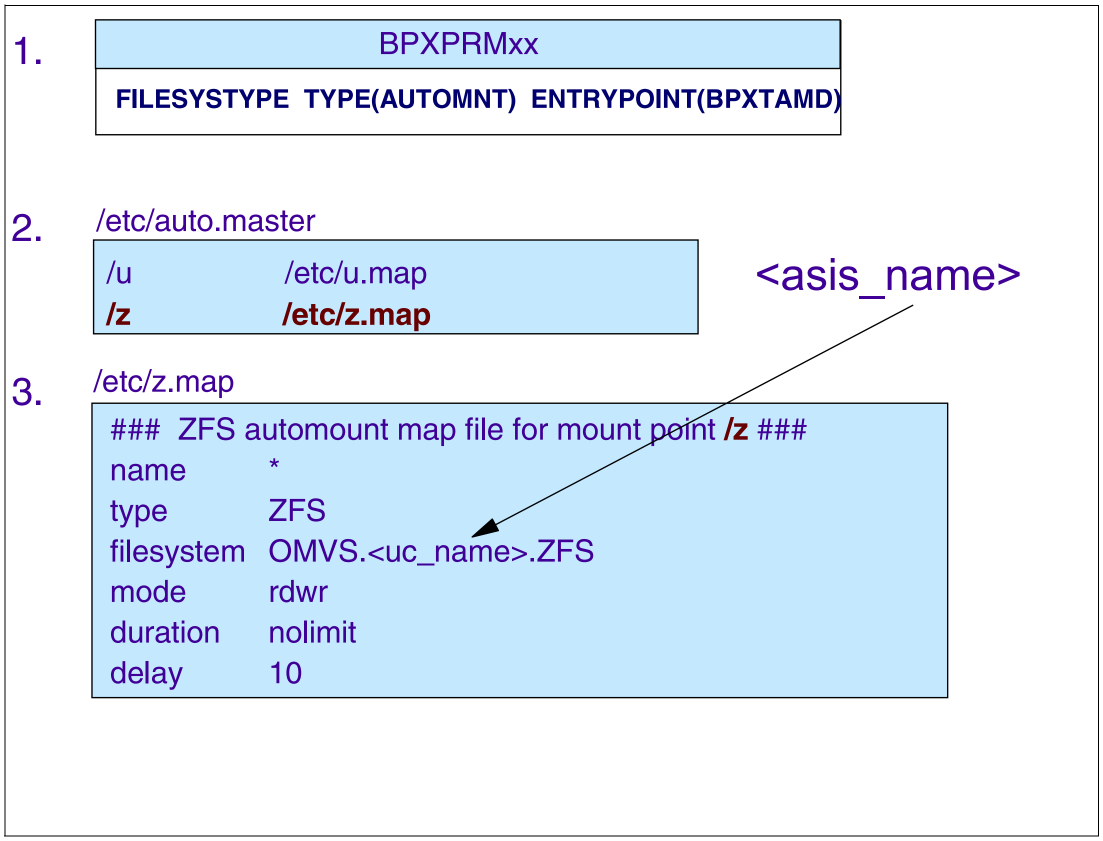
```
#> cd /
#> mkdir z
```

One of the main concerns when creating zFS file systems is where they should be mounted in the z/OS UNIX hierarchy. If you choose to add to an existing automount policy, you can modify it by doing the following:

- Adding a new entry in the auto.master file
- Creating a new map file for the new mount point in the auto.master file

Note: The pathname of the managed directory is used as a filesystem name prefixed with `*AMD`.

7.62 Automount policy for zFS



```
FILESYSTYPE  TYPE(AUTOMNT)  ENTRYPOINT(BPXTAMD)
```

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

```
###  ZFS automount map file for mount point /z ###
name         *
type         ZFS
filesystem   OMVS.<uc_name>.ZFS
mode         rdwr
duration     nolimit
delay        10
```

Figure 7-62   Defining the automount policy for zFS file systems

The file system name in the z.map file specifies the file systems that are to be mounted by the automount facility. In our previous examples, we created the multi-file aggregate OMVS.MUL02.ZFS. We defined the following file systems in the aggregate:

```
ROGERS @ SC43:>zfsadm lsfs -a OMVS.MUL02.ZFS -fast
OMVS.M02A.ZFS
OMVS.M02D.ZFS
OMVS.M02B.ZFS
OMVS.m02c.ZFS
```

Using <uc_name>

The <uc_name> variable is used to convert the name being looked up to uppercase. Whenever this variable is encountered it is replaced by the name being looked up. A directory with the looked-up name is created, and used as a mount point for the file system to be mounted. When creating a map file, the <uc_name> variable can be used to replace any level qualifier in the data set name.

7.63 Automount of a zFS file system

- Users or programs reference a file system

```
ROGERS @ SC43:>cd /z/m02a
ROGERS @ SC43:>/z/m02a>ls -al
total 336
drwxr-xr-x   9 HERING   SYS1         736 Apr  2 01:43 .
drwxr-xr-x   9 HERING   SYS1         736 Apr  2 01:43 ..
-rwxr-xr-x   1 HERING   SYS1        1869 Mar 14 16:27 .profile
-rwxr-xr-x   1 HERING   SYS1         689 Mar 14 16:29 .setup
-rw-------   1 HERING   SYS1        3033 Apr  1 15:29 .sh_history
drwxr-xr-x   2 HERING   SYS1         256 Mar 15 16:08 bin
drwx------   4 RC43     SYS1        1088 Apr  4 00:35 test
-r--r--r--   1 RC43     SYS1         147 Mar 17 14:41 test.extattr
drwx------   2 RC43     SYS1         256 Mar 31 15:29 test1
lrwxrwxrwx   1 RC43     SYS1          15 Apr  4 00:35 test1.sl ->
/u/hering/test
1
drwxr-xr-x   2 HERING   SYS1         256 Mar 14 14:58 test2
drwxr-xr-x   2 HERING   SYS1         256 Mar 19 00:43 test3
```

Figure 7-63 Example of mounting a zFS file system with automount

A file system is mounted by the automount facility when any user issues the commands shown in Figure 7-63:

```
ROGERS @ SC43:>cd /z/m02a
ROGERS @ SC43:>/z/m02a>ls -al
```

Once the file system is mounted, the response to the command **`ls -al`** that caused the file system to be mounted is displayed for the user, as shown in the figure.

7.64 zFS file systems mounted (automount)

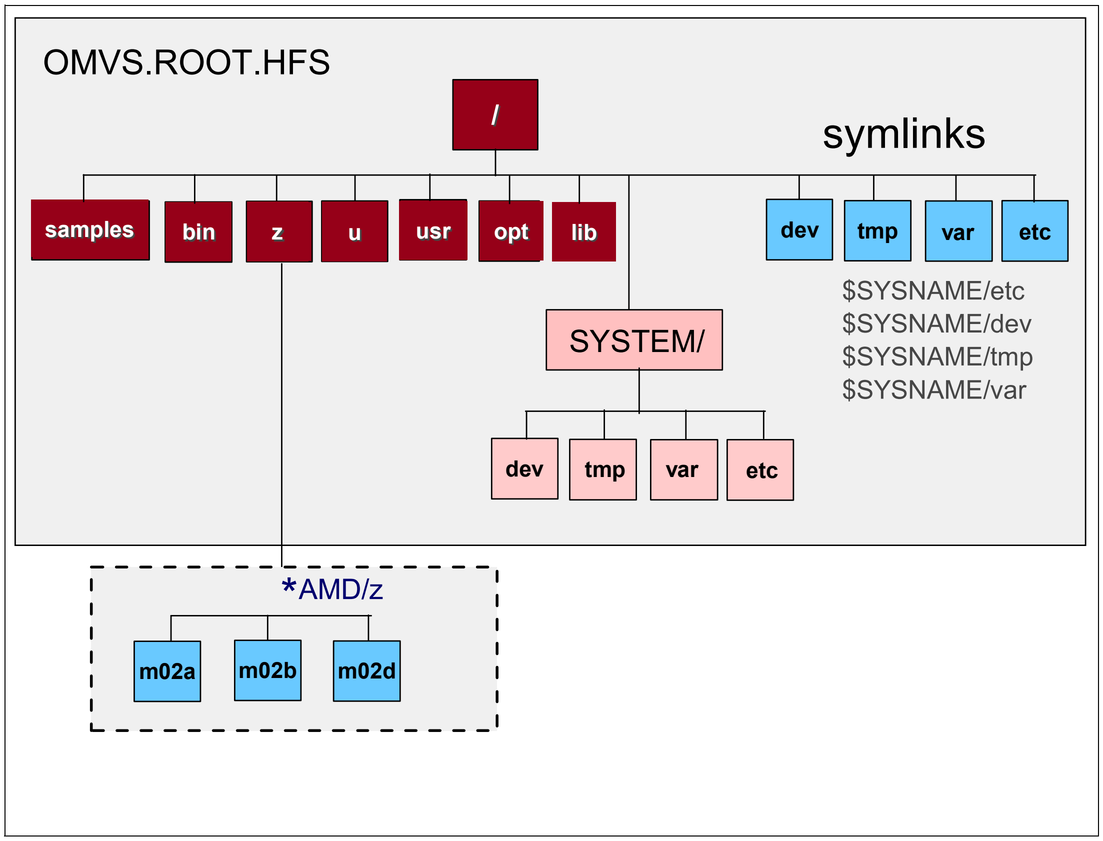
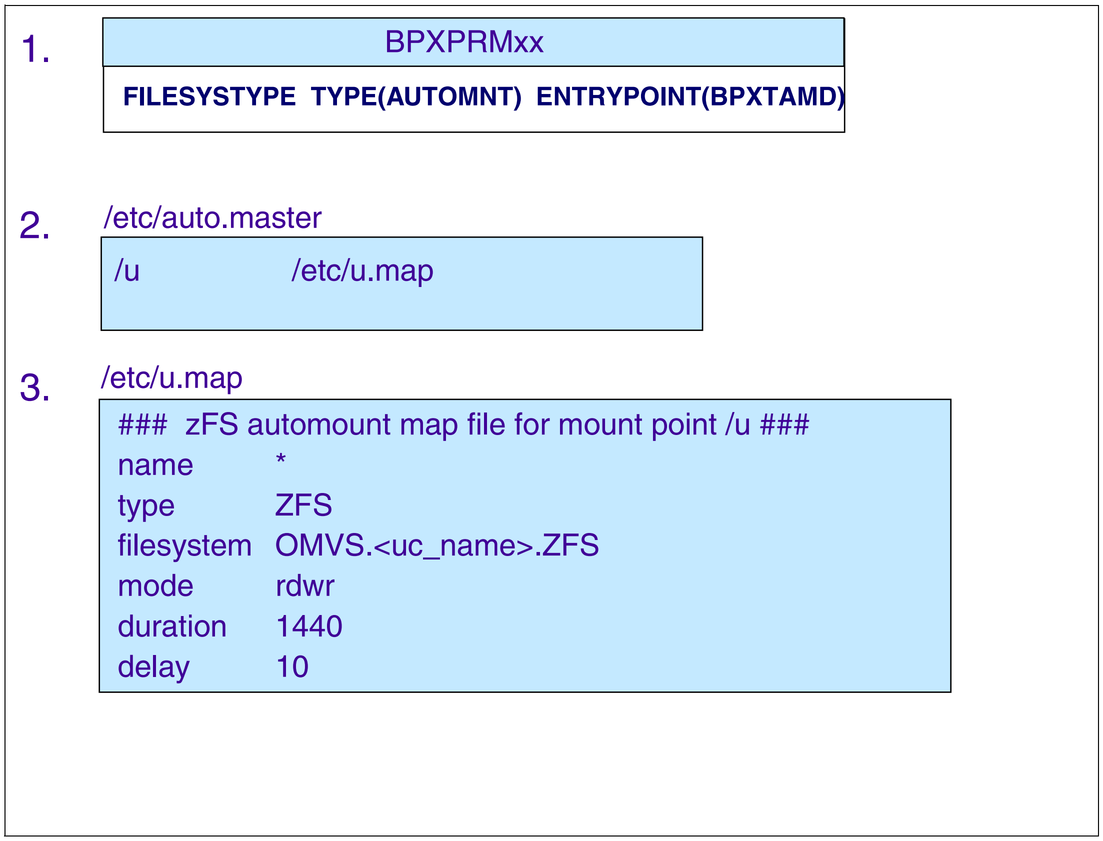


Figure 7-64 Example of the automount pseudo directories for mount of file systems

When all the file systems have been accessed, Figure 7-64 shows that *AMD/z is managing the /z directory and three file systems have been mounted on directories m02a, m02b, and mo2d.

7.65 Automount for users with /u



1. BPXPRMxx

```
FILESYSTYPE  TYPE(AUTOMNT)  ENTRYPOINT(BPXTAMD)
```

2. /etc/auto.master

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

3. /etc/u.map

```
###  zFS automount map file for mount point /u ###
name          *
type          ZFS
filesystem    OMVS.<uc_name>.ZFS
mode          rdwr
duration      1440
delay         10
```

Figure 7-65 Using the automount policy to mount user file systems with zFS

You can change your current automount facility to manage your user file systems as zFS file systems. Then, any zFS file system that was previously mounted will be mounted by the automount facility when the mount point is referenced by a user. With the automount facility, you do not need to mount user file systems at system IPL.

You also do not need to request that operators perform mounts for other file systems. In addition, the facility simplifies the addition of new users because you do not need to update your parmlib definitions for mounts. You can establish a simple automount policy to manage user home directories, as shown in the figure.

Following are several advantages of using automount with a zFS multi-file aggregate containing the file systems for users:

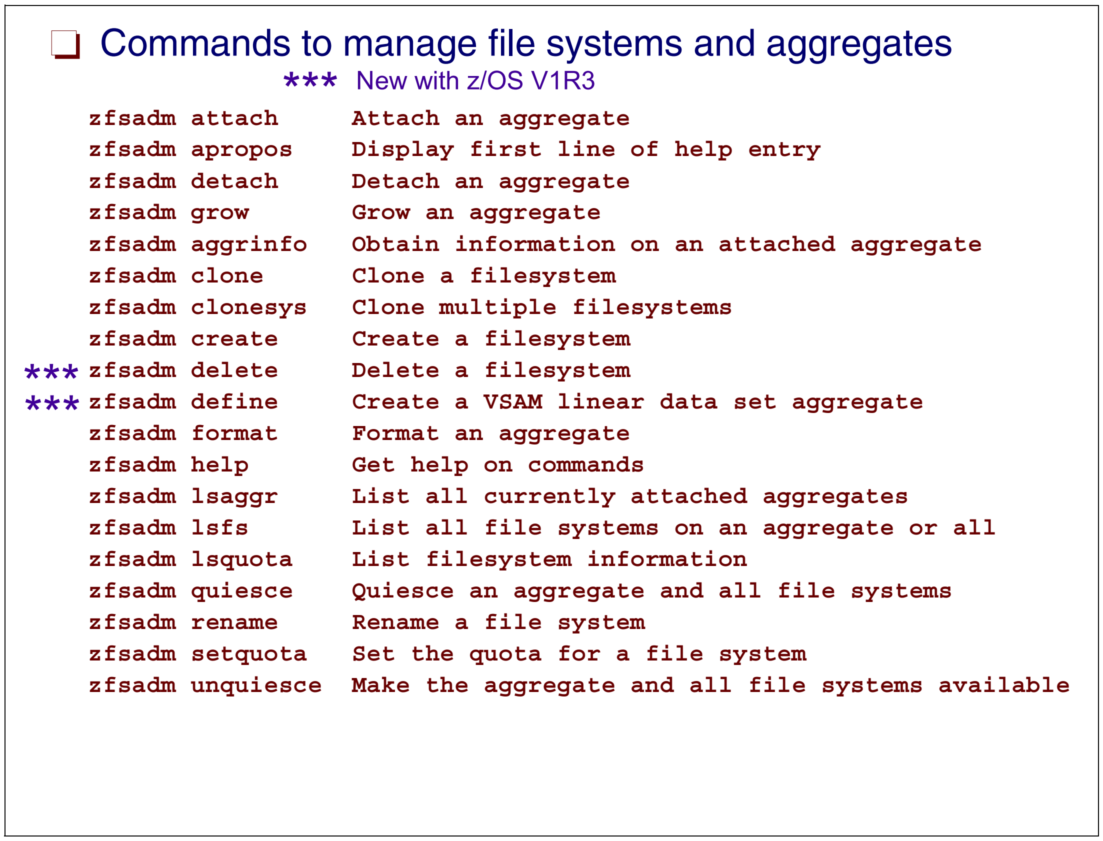
- By defining a multi-file aggregate for the z/OS UNIX users, all the users can share the allocated space in the aggregate. This eliminates having to increase space for individual users in their file systems. If all the space is used in the aggregate, you can issue the **`zfsadm grow`** command to increase the space for all users.
- As users tend to have very different needs for their amount of space, you do not waste disk space because it is not needed to provide a separate contingent of disk space for each individual user.

Define a larger logical amount of space for a user using the **`zfsadm setquota`** command only if that user has used up his allowed quota. No physical change is necessary if there is still enough space available in the aggregate.

Figure 7-65 shows the steps required to define the automount environment.

1. Have the automount FILESYSTYPE active, either by having the statement in the active BPXPRMxx member during IPL as shown or by using the **SETOMVS RESET=(xx)** command to do it dynamically.
2. The auto.master file in /etc must contain all mount points that should be managed by automount.
3. A map file describes the automount policy for a specific mount point. If you choose to include a system name, you can specify &SYSNAME. instead of <system>, which exploits the new system symbols support of automount.

7.66 zfsadm commands



Commands to manage file systems and aggregates

*** New with z/OS V1R3

zfsadm attach	Attach an aggregate
zfsadm apropos	Display first line of help entry
zfsadm detach	Detach an aggregate
zfsadm grow	Grow an aggregate
zfsadm aggrinfo	Obtain information on an attached aggregate
zfsadm clone	Clone a filesystem
zfsadm clonesys	Clone multiple filesystems
zfsadm create	Create a filesystem
*** zfsadm delete	Delete a filesystem
*** zfsadm define	Create a VSAM linear data set aggregate
zfsadm format	Format an aggregate
zfsadm help	Get help on commands
zfsadm lsaggr	List all currently attached aggregates
zfsadm lsfs	List all file systems on an aggregate or all
zfsadm lsquota	List filesystem information
zfsadm quiesce	Quiesce an aggregate and all file systems
zfsadm rename	Rename a file system
zfsadm setquota	Set the quota for a file system
zfsadm unquiesce	Make the aggregate and all file systems available

Figure 7-66 zFS commands

zFS provides utility programs and z/OS UNIX commands to assist in the customization of the aggregates and file systems. These utilities and commands should be used only by system administrators.

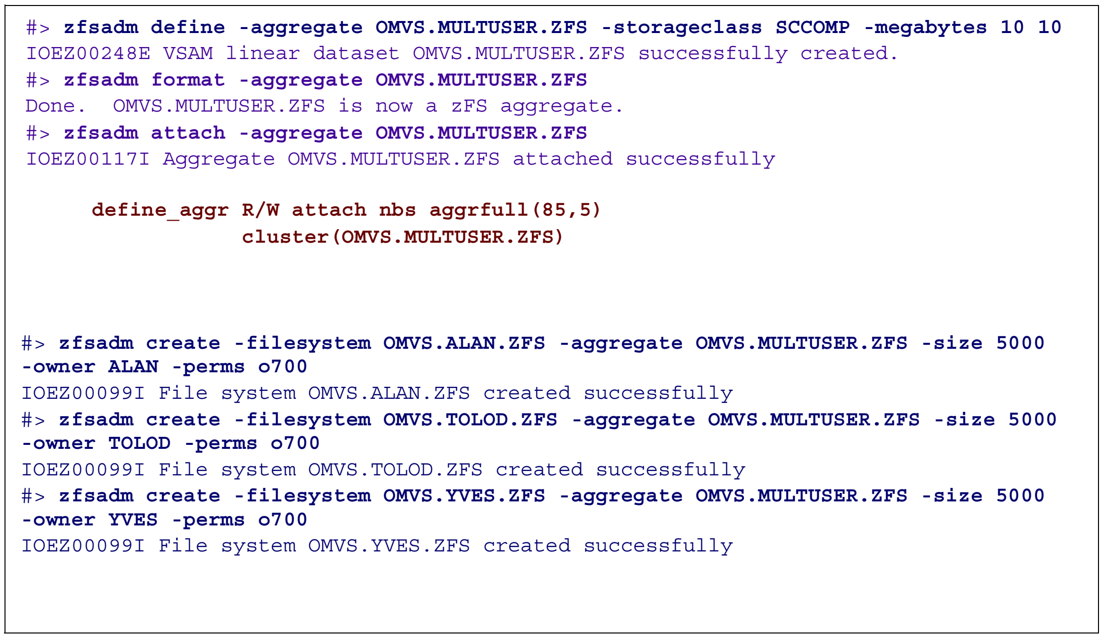
The **zfsadm** command and the IOEZADM utility program can be used to manage file systems and aggregates.

The **zfsadm** command can be run as a UNIX shell command from:

- A z/OS UNIX system services shell (OMVS) or a (z/OS UNIX) telnet session
- A batch job using the BPXBATCH utility program
- TSO foreground or in batch mode using z/OS UNIX APIs (SYSCALL commands, Callable Services)

zFS file systems can be created using JCL, or by using the **zfsadm** command. Figure 7-66 shows the **zfsadm** command with its subcommands.

7.67 Defining the automount environment



```
#> zfsadm define -aggregate OMVS.MULTUSER.ZFS -storageclass SCCOMP -megabytes 10 10
IOEZ00248E VSAM linear dataset OMVS.MULTUSER.ZFS successfully created.
#> zfsadm format -aggregate OMVS.MULTUSER.ZFS
Done.  OMVS.MULTUSER.ZFS is now a zFS aggregate.
#> zfsadm attach -aggregate OMVS.MULTUSER.ZFS
IOEZ00117I Aggregate OMVS.MULTUSER.ZFS attached successfully

     define_aggr R/W attach nbs aggrfull(85,5)
                 cluster(OMVS.MULTUSER.ZFS)



#> zfsadm create -filesystem OMVS.ALAN.ZFS -aggregate OMVS.MULTUSER.ZFS -size 5000
-owner ALAN -perms o700
IOEZ00099I File system OMVS.ALAN.ZFS created successfully
#> zfsadm create -filesystem OMVS.TOLOD.ZFS -aggregate OMVS.MULTUSER.ZFS -size 5000
-owner TOLOD -perms o700
IOEZ00099I File system OMVS.TOLOD.ZFS created successfully
#> zfsadm create -filesystem OMVS.YVES.ZFS -aggregate OMVS.MULTUSER.ZFS -size 5000
-owner YVES -perms o700
IOEZ00099I File system OMVS.YVES.ZFS created successfully
```

Figure 7-67 Defining the automount environment for zFS file system users

First you define, format, and attach the aggregate. By defining a multi-file aggregate for the z/OS UNIX users, all the users can share the allocated space in the aggregate. This eliminates having to increase space for individual users in their file systems. Do it from the OMVS shell, as follows:

```
#> zfsadm define -aggregate OMVS.MULTUSER.ZFS -storageclass SCCOMP -megabytes 10 10
IOEZ00248E VSAM linear dataset OMVS.MULTUSER.ZFS successfully created.
#> zfsadm format -aggregate OMVS.MULTUSER.ZFS
Done.  OMVS.MULTUSER.ZFS is now a zFS aggregate.
#> zfsadm attach -aggregate OMVS.MULTUSER.ZFS
IOEZ00117I Aggregate OMVS.MULTUSER.ZFS attached successfully.
```

In parallel, we added a `define_aggr` statement to the IOEFSPRM parameter file, shown in Figure 7-67. This attach of the aggregate is only done during a restart of the zFS address space.

To create the user's individual file systems, the figure shows how to create some user file systems with a sum of allowed quotas (15 MB) that is already more than the initial size of the whole aggregate as it has been defined (10 MB). This demonstrates the flexibility of this way of defining automount user file systems.

```
#> zfsadm create -filesystem OMVS.ALAN.ZFS -aggregate OMVS.MULTUSER.ZFS -size 5000
-owner ALAN -perms o700
IOEZ00099I File system OMVS.ALAN.ZFS created successfully
#> zfsadm create -filesystem OMVS.TOLOD.ZFS -aggregate OMVS.MULTUSER.ZFS -size 5000
-owner TOLOD -perms o700
IOEZ00099I File system OMVS.TOLOD.ZFS created successfully
#> zfsadm create -filesystem OMVS.YVES.ZFS -aggregate OMVS.MULTUSER.ZFS -size 5000
-owner YVES -perms o700
IOEZ00099I File system OMVS.YVES.ZFS created successfully
```

7.68 Automount for users /u

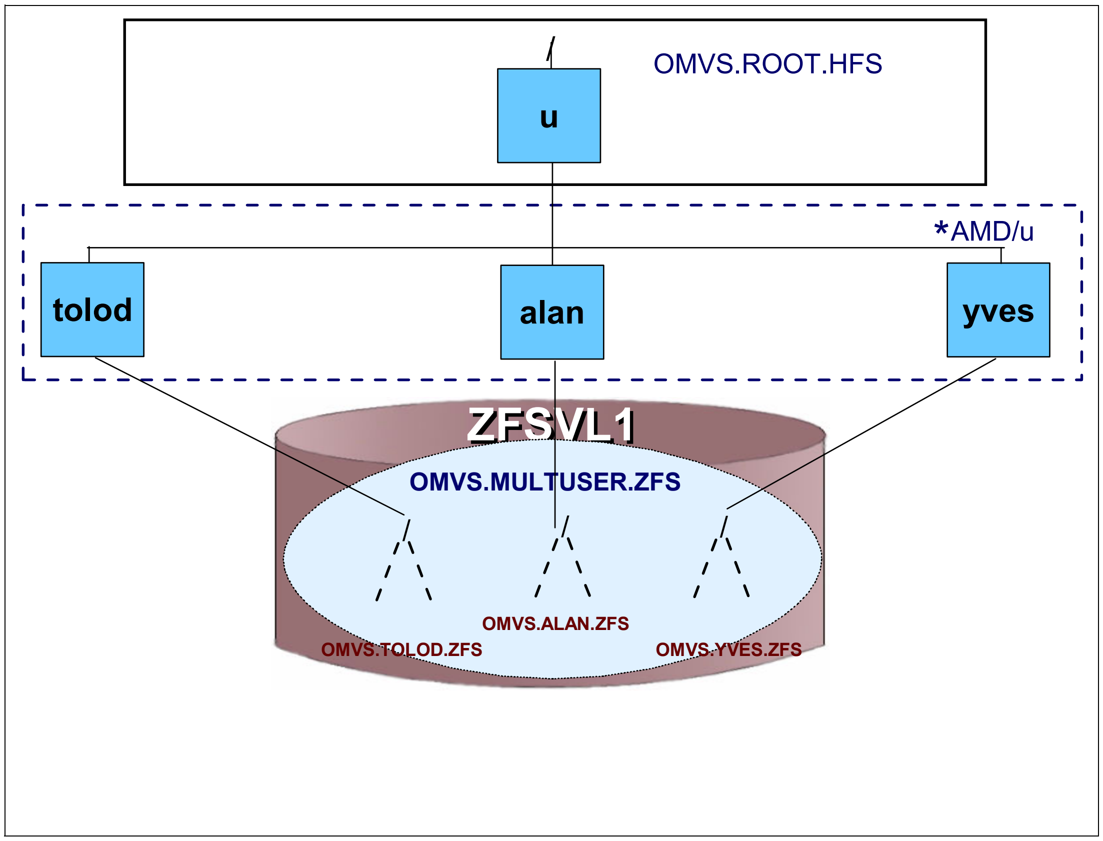


Figure 7-68 Example of user file systems sharing space while mounted in a zFS aggregate

Figure 7-68 shows the z/OS UNIX automount file structure and the zFS multi-file aggregate with the user file systems that it contains. The three users, ALAN, YVES, and TOLOD have their file systems mounted by the automount facility. The file systems are sharing the space defined in the aggregate, OMVS.MULTUSER.ZFS.

7.69 File sharing in a sysplex

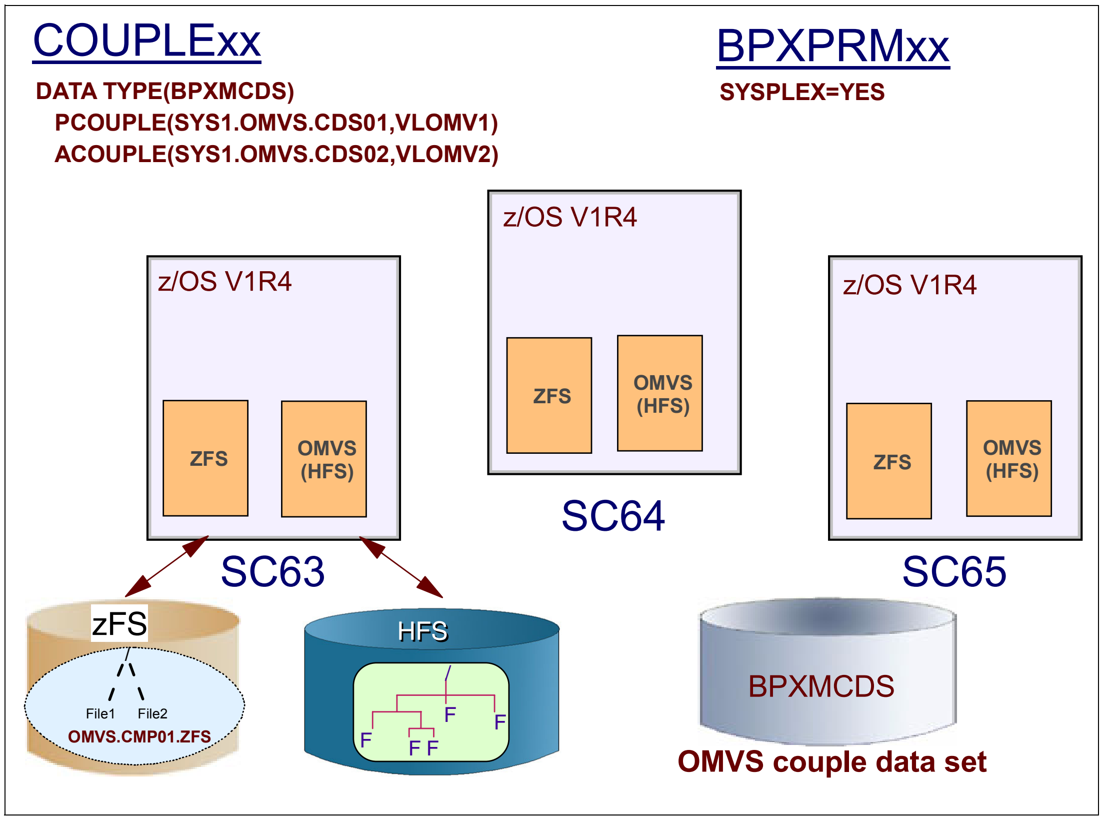


Figure 7-69 File sharing in a sysplex environment

Before OS/390 UNIX V2R9, users could have read/write access only to data in file systems mounted on their own system. With shared HFS, users have greater access to the file systems in a sysplex. They have read/write access to file systems that are mounted on other systems.

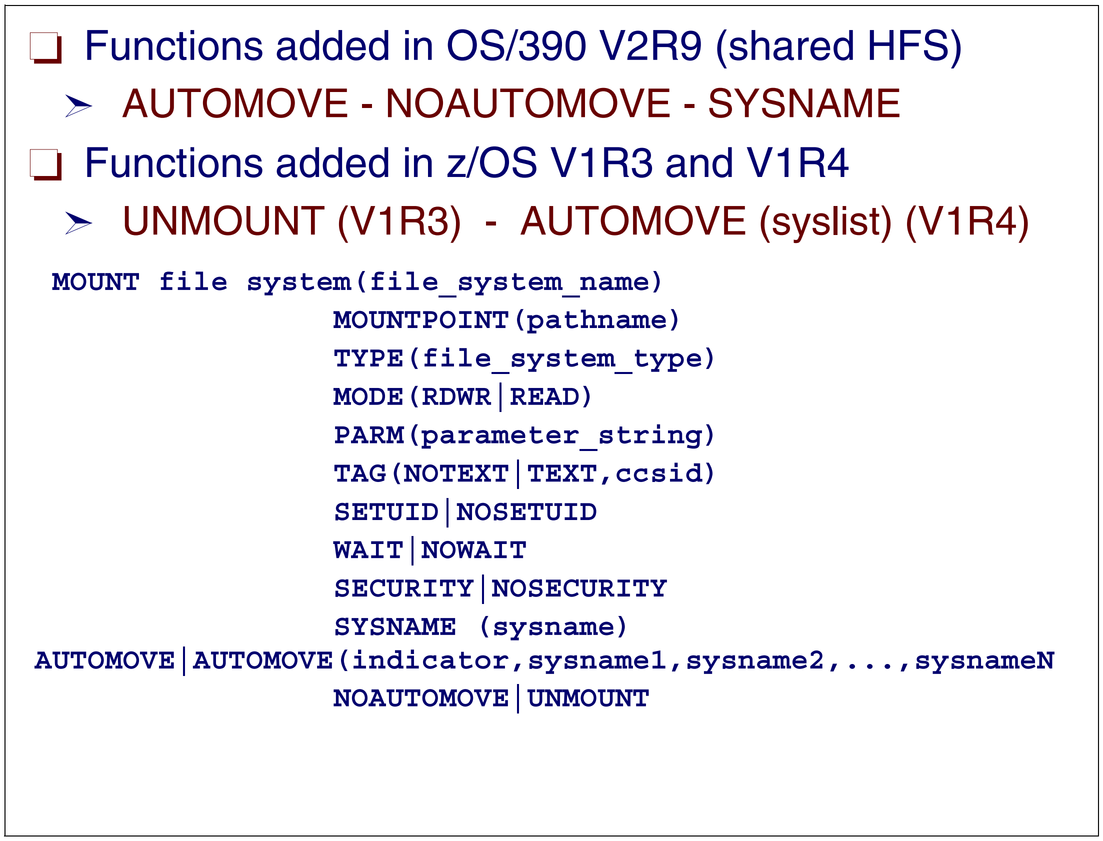
With shared HFS, all file systems that are mounted by a system participating in shared HFS are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a OS/390 UNIX V2R9 sysplex that consists of three systems, SC63, SC64, and SC65:

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

The CDS must be defined in the COUPLExx parmlib member. Shared HFS must be defined in the BPXPRMxx parmlib member.

7.70 Mounting file systems (HFS - zFS)



- ❑ Functions added in OS/390 V2R9 (shared HFS)
  - ➢ AUTOMOVE - NOAUTOMOVE - SYSNAME
- ❑ Functions added in z/OS V1R3 and V1R4
  - ➢ UNMOUNT (V1R3) - AUTOMOVE (syslist) (V1R4)

```
 MOUNT file system(file_system_name)
                MOUNTPOINT(pathname)
                TYPE(file_system_type)
                MODE(RDWR|READ)
                PARM(parameter_string)
                TAG(NOTEXT|TEXT,ccsid)
                SETUID|NOSETUID
                WAIT|NOWAIT
                SECURITY|NOSECURITY
                SYSNAME (sysname)
AUTOMOVE|AUTOMOVE(indicator,sysname1,sysname2,...,sysnameN
                NOAUTOMOVE|UNMOUNT
```

Figure 7-70 Options when mounting file systems

These parameters that have been added to the MOUNT processing commands apply only in a sysplex where systems are participating in shared HFS or shared zFS. They indicate what happens if the system that owns a file system goes down.

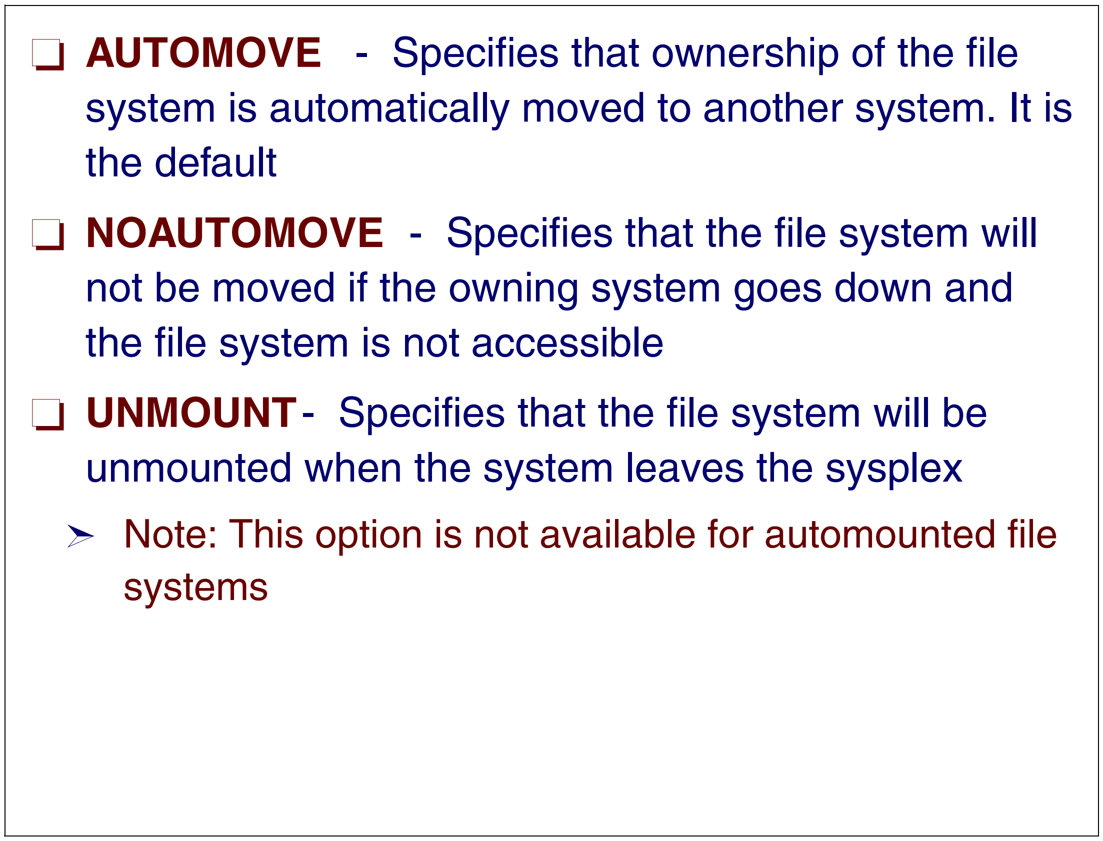
AUTOMOVE(indicator,sysname1,sysname2,...,sysnameN)

This new AUTOMOVE feature of z/OS v1R4 allows for a list of systems to be included or excluded (indicated by the indicator field) from being AUTOMOVEd. The `include` indicator and system list provides an ordered list of systems that should be moved to if the file system's owning system should leave the sysplex. The `exclude` indicator and system list provides a list of systems that the file system should not be moved to if the file system's owning system should leave the sysplex.

SYSNAME

For systems participating in shared HFS, SYSNAME specifies the particular system on which a mount should be performed. This system will then become the owner of the file system mounted. This system must be IPLed with SYSPLEX(YES). IBM recommends that you specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

7.71 MOUNT command options



- **AUTOMOVE** - Specifies that ownership of the file system is automatically moved to another system. It is the default
- **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
  - Note: This option is not available for automounted file systems

Figure 7-71 Mount command options for the shared sysplex environment

AUTOMOVE

When AUTOMOVE is specified for a file system and the file system's owner goes down, AUTOMOVE indicates that ownership of the file system can be automatically moved to another system participating in shared sysplex HFS or zFS. AUTOMOVE is the default.

NOAUTOMOVE

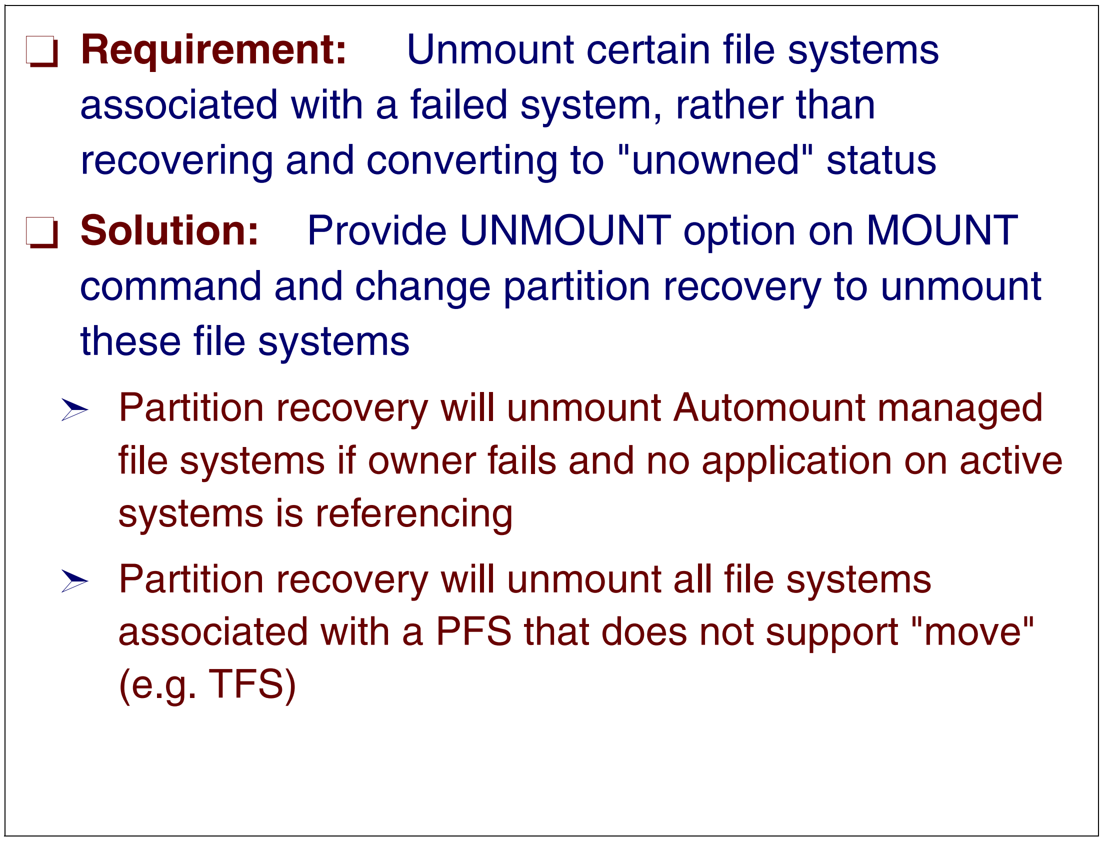
When NOAUTOMOVE is specified for a file system, this indicates that ownership should not be moved to another system participating in shared HFS if the file system's owner should crash.

Note: AUTOMOVE is not supported for zFS file systems mounted in a multi-system mode aggregate. They must be mounted NOAUTOMOVE.

UNMOUNT

When UNMOUNT is specified for a file system, this indicates that the file system will not be moved and will be unmounted if the file system's owner should crash. This file system and any file systems mounted within its subtree will be unmounted.

7.72 UNMOUNT option



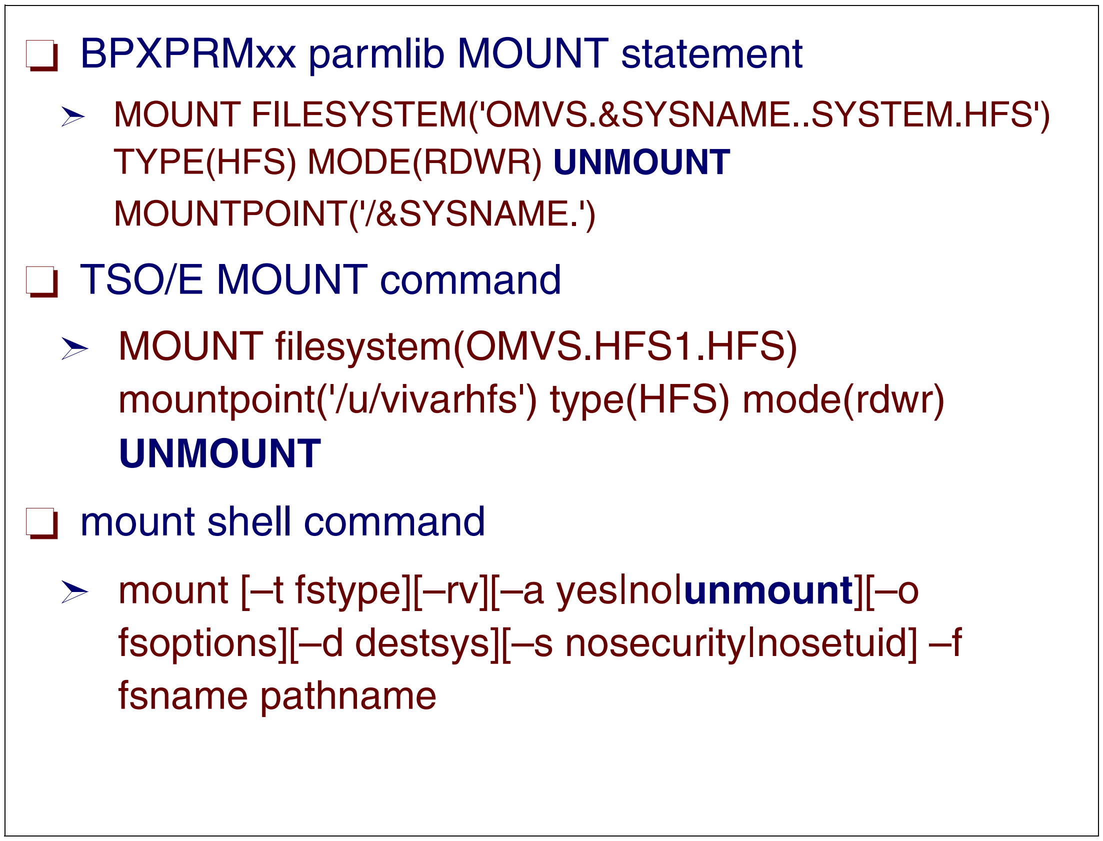
- **Requirement:** Unmount certain file systems associated with a failed system, rather than recovering and converting to "unowned" status
- **Solution:** Provide UNMOUNT option on MOUNT command and change partition recovery to unmount these file systems
  - Partition recovery will unmount Automount managed file systems if owner fails and no application on active systems is referencing
  - Partition recovery will unmount all file systems associated with a PFS that does not support "move" (e.g. TFS)

Figure 7-72 UNMOUNT option on mount command

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

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

7.73 UNMOUNT option support



- BPXPRMxx parmlib MOUNT statement
  - MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS') TYPE(HFS) MODE(RDWR) **UNMOUNT** MOUNTPOINT('/&SYSNAME.')
- TSO/E MOUNT command
  - MOUNT filesystem(OMVS.HFS1.HFS) mountpoint('/u/vivarhfs') type(HFS) mode(rdwr) **UNMOUNT**
- mount shell command
  - mount [–t fstype][–rv][–a yes|no|**unmount**][–o fsoptions][–d destsys][–s nosecurity|nosetuid] –f fsname pathname

Figure 7-73 UNMOUNT option support for activation

BPXPRMxx parmlib MOUNT statement

The `AUTOMOVE | NOAUTOMOVE | UNMOUNT` parameters on ROOT and MOUNT statements indicate what happens to the file system if the system that owns that file system goes down. `UNMOUNT` specifies that the file system will be unmounted when the system leaves the sysplex. This option is not available for automounted file systems. `AUTOMOVE` is the default.

TSO/E MOUNT command

The options are: `AUTOMOVE | NOAUTOMOVE | UNMOUNT`. The `AUTOMOVE | NOAUTOMOVE` parameters apply only in a sysplex where systems are participating in shared HFS. They indicate what happens if the system that owns a file system goes down. The default setting is `AUTOMOVE`.

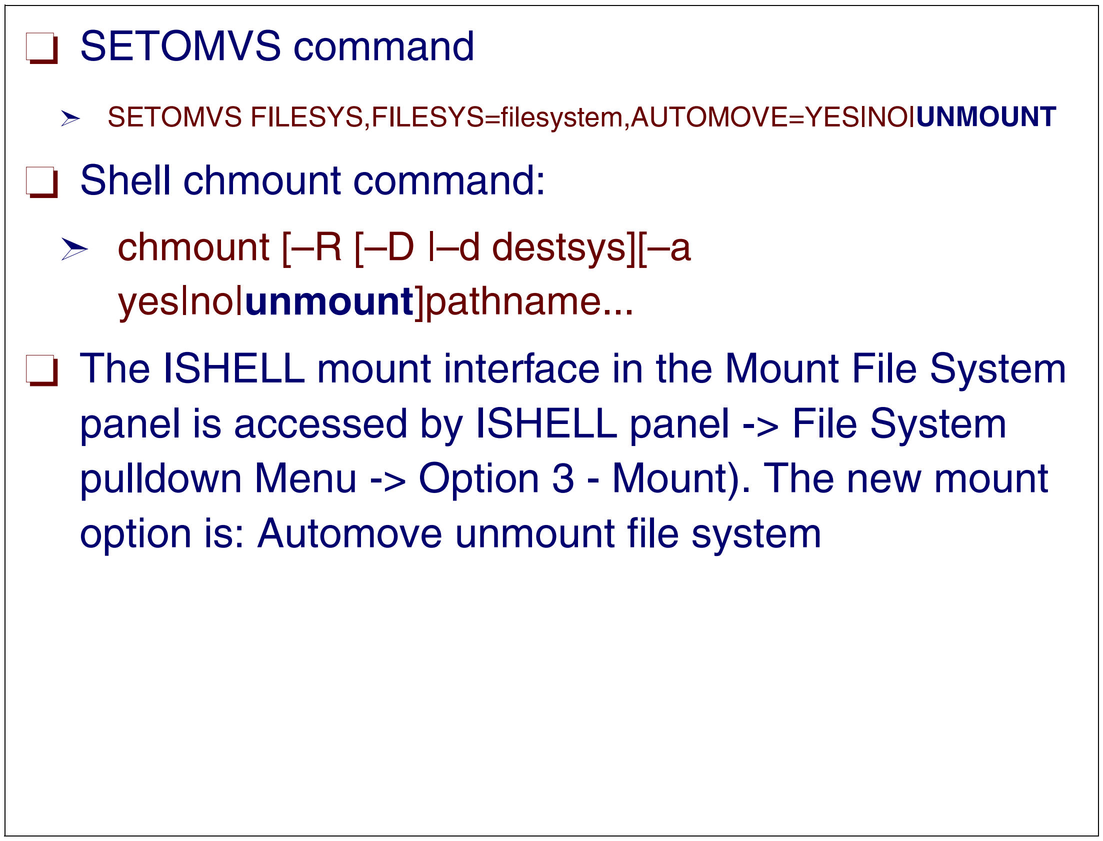
The `mount` shell command

The **`mount`** shell command, located in /usr/sbin, is used to mount a file system or list all mounts over a file system. A mount user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Specify unmount to allow a specified file system (and any file systems mounted on it) to be unmounted when its owning system leaves the sysplex due to a system outage.

The options are: `-a yes|no|unmount`. The default is `yes`.

7.74 UNMOUNT option support (2)

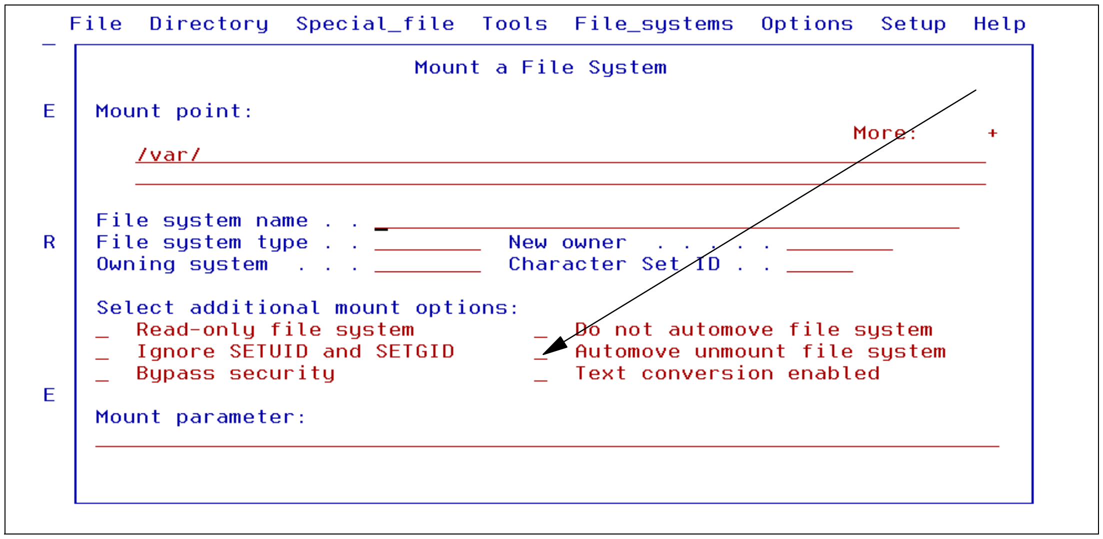


- SETOMVS command
  - SETOMVS FILESYS,FILESYS=filesystem,AUTOMOVE=YES|NO|**UNMOUNT**
- Shell chmount command:
  - chmount [–R [–D |–d destsys][–a yes|no|**unmount**]pathname...
- The ISHELL mount interface in the Mount File System panel is accessed by ISHELL panel -> File System pulldown Menu -> Option 3 - Mount). The new mount option is: Automove unmount file system

Figure 7-74 UNMOUNT options for activation

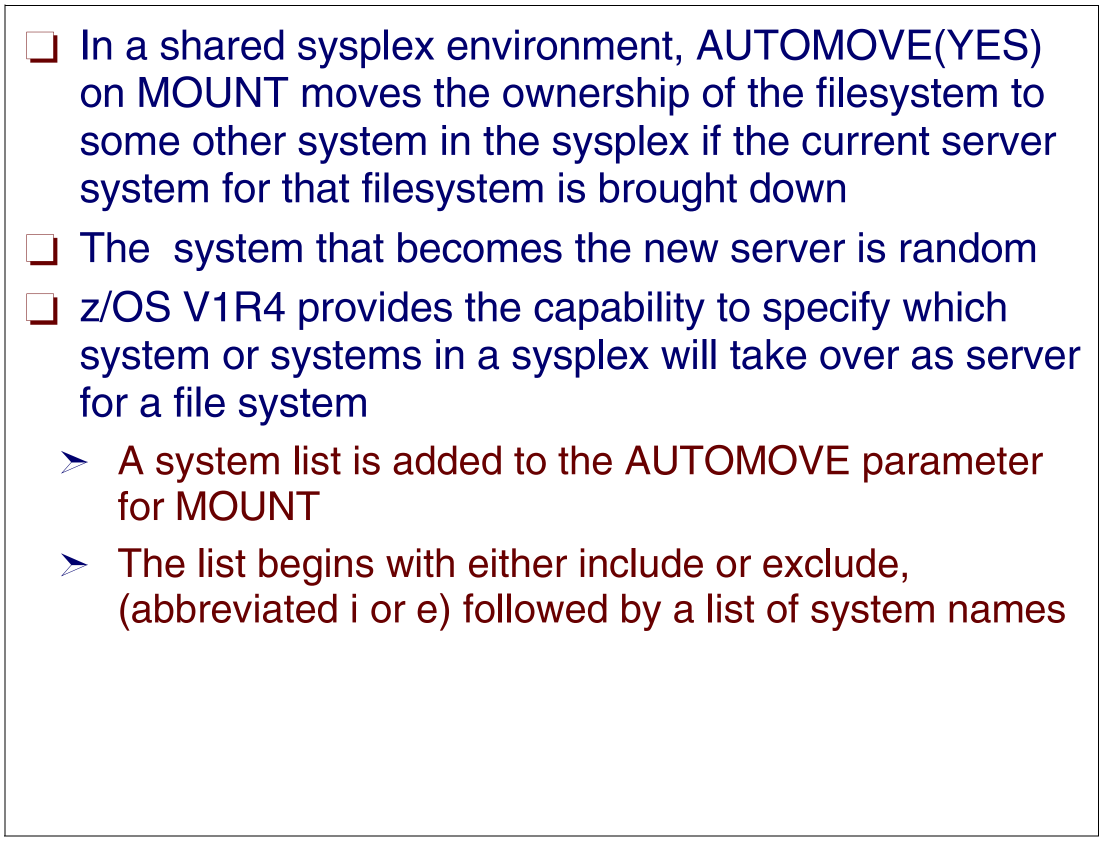
The `UNMOUNT` option can be used on the following commands:

- The **SETOMVS** command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex.
- You can use the **chmount** command from the shell.
- The ISHELL can be used to set the UNMOUNT option as follows:



```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 _
                        Mount a File System

 E   Mount point:
                                                            More:     +
       /var/
       ______________________________________________________________

     File system name . . _______________________________________
 R   File system type . . ________  New owner  . . . . . . ________
     Owning system   . . . ________  Character Set ID . . _____

     Select additional mount options:
     _  Read-only file system          _  Do not automove file system
     _  Ignore SETUID and SETGID       _  Automove unmount file system
     _  Bypass security                _  Text conversion enabled
 E
     Mount parameter:
     ____________________________________________________________________
```

7.75 AUTOMOVE system list (syslist)



- In a shared sysplex environment, AUTOMOVE(YES) on MOUNT moves the ownership of the filesystem to some other system in the sysplex if the current server system for that filesystem is brought down
- The system that becomes the new server is random
- z/OS V1R4 provides the capability to specify which system or systems in a sysplex will take over as server for a file system
  - A system list is added to the AUTOMOVE parameter for MOUNT
  - The list begins with either include or exclude, (abbreviated i or e) followed by a list of system names

Figure 7-75 AUTOMOVE system list specifications

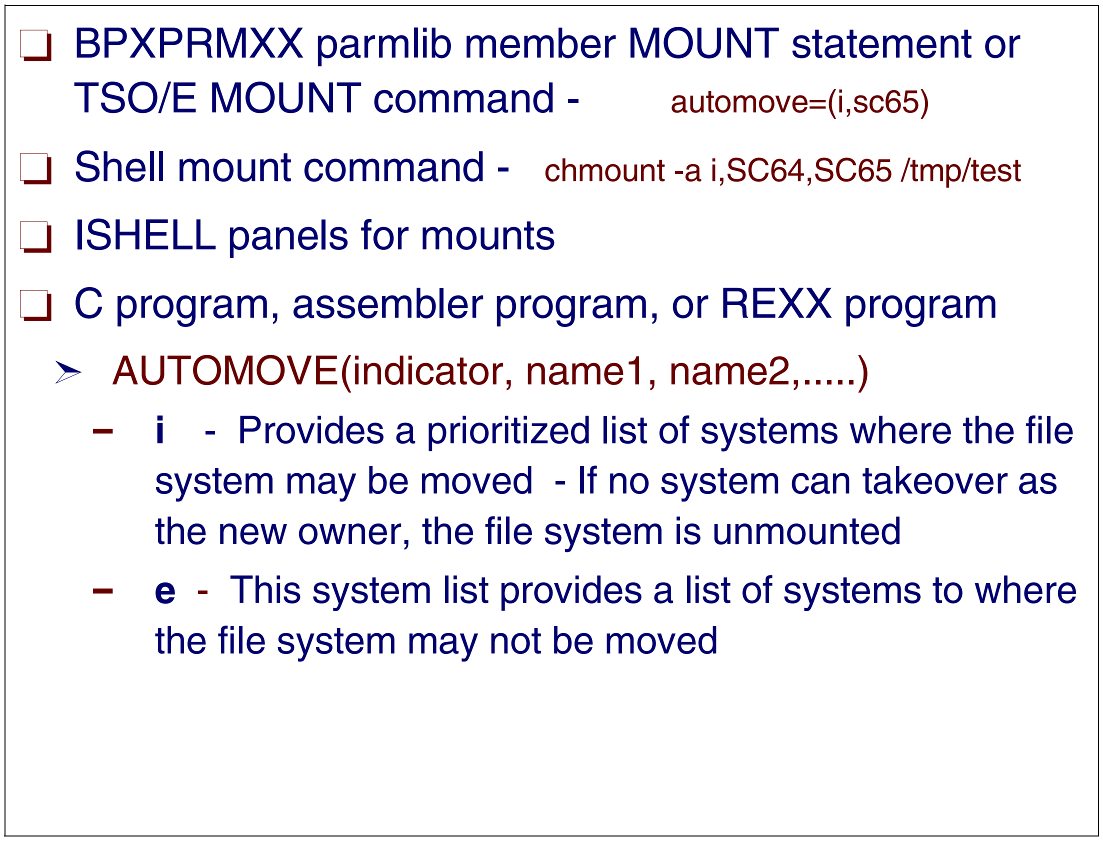
When mounting file systems, you can specify an automove system list to indicate where the file system should or should not be moved when a system leaves the sysplex. Previously, on a system failure, with more than one system capable of taking over ownership, it was random which system was the new owner.

The automove system list can be specified in many different ways to automove a file system. The list begins with an indicator to either *include* or *exclude* (abbreviated as **i** or **e**) followed by a list of system names. Specify the indicator as follows:

i Use this indicator with a prioritized list of systems to which the file system may be moved if the owning system goes down. If none of the systems specified in the list can take over as the new owner, the file system will be unmounted.

e Use this indicator with a list of systems to which the file system may *not* be moved.

7.76 AUTOMOVE parameters for mounts



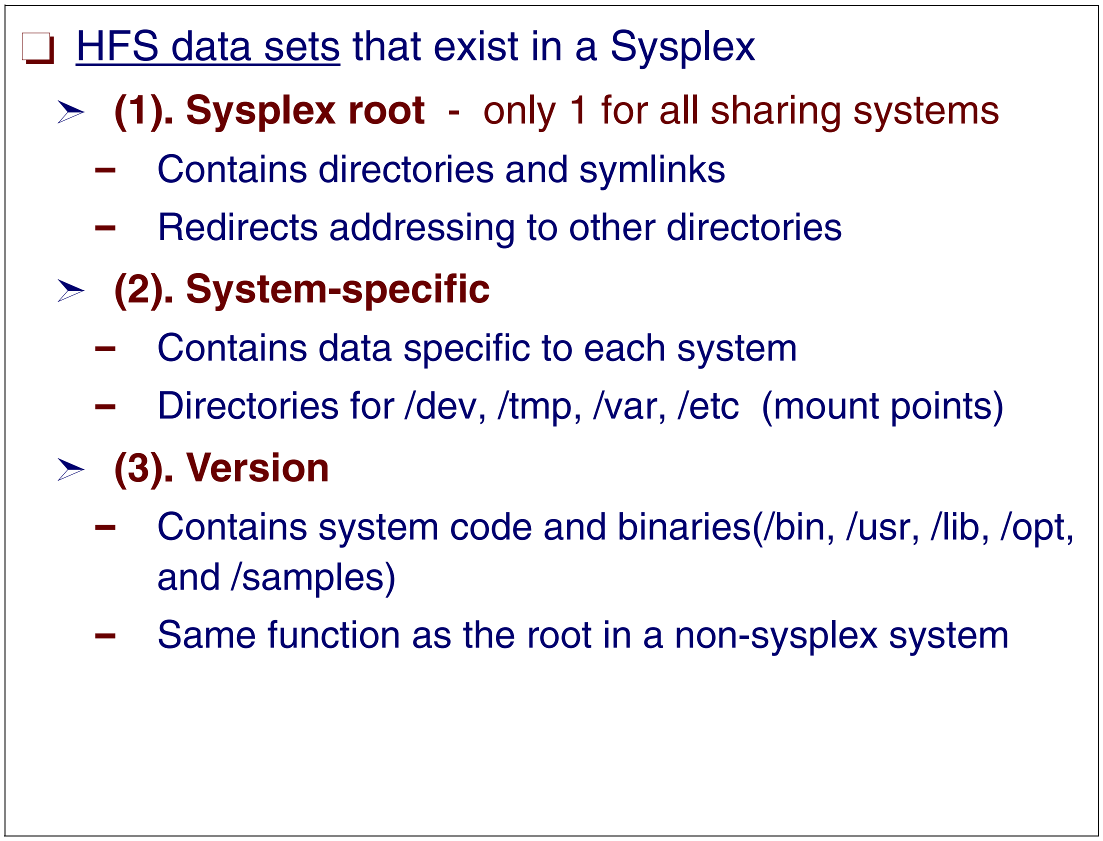
- BPXPRMXX parmlib member MOUNT statement or TSO/E MOUNT command - automove=(i,sc65)
- Shell mount command - chmount -a i,SC64,SC65 /tmp/test
- ISHELL panels for mounts
- C program, assembler program, or REXX program
  - AUTOMOVE(indicator, name1, name2,.....)
    - **i** - Provides a prioritized list of systems where the file system may be moved - If no system can takeover as the new owner, the file system is unmounted
    - **e** - This system list provides a list of systems to where the file system may not be moved

Figure 7-76 AUTOMOVE specifications for system lists

When mounting file systems in the sysplex, you can specify a prioritized automove system list to indicate where the file system should or should not to moved to when the owning system leaves the sysplex. There are different ways to specify the automove system list, as follows:

- 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.

7.77 Shared file systems in a sysplex



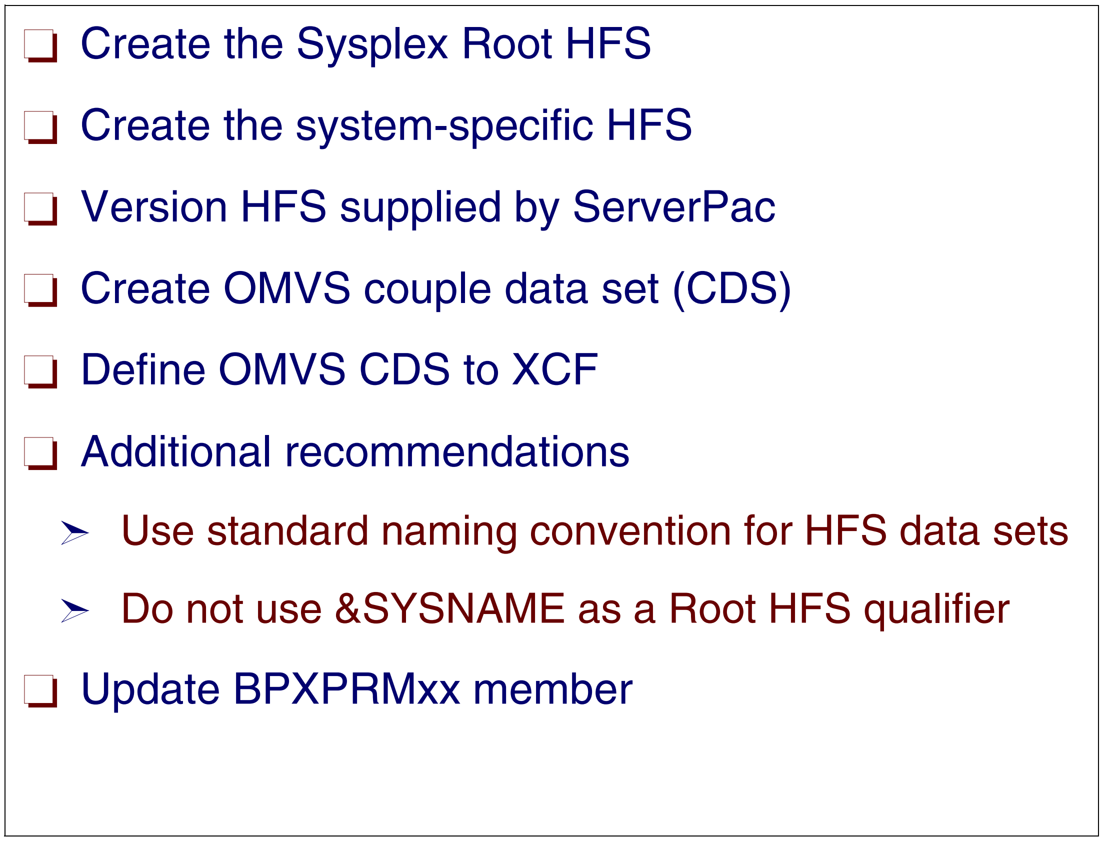
- HFS data sets that exist in a Sysplex
  - **(1). Sysplex root** - only 1 for all sharing systems
    - Contains directories and symlinks
    - Redirects addressing to other directories
  - **(2). System-specific**
    - Contains data specific to each system
    - Directories for /dev, /tmp, /var, /etc (mount points)
  - **(3). Version**
    - Contains system code and binaries(/bin, /usr, /lib, /opt, and /samples)
    - Same function as the root in a non-sysplex system

Figure 7-77 Shared sysplex environment file systems

The three file systems in Figure 7-77 are as follows:

1. This is the sysplex root HFS data set, which was created by running the BPXISYSR job. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.
2. This is the system-specific HFS data set, which was created by running the BPXISYSS job. It must be mounted read-write. NOAUTOMOVE is specified because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPOINT statement `/&SYSNAME.` will resolve to /SC64 during parmlib processing. This mount point is created dynamically at system initialization.
3. This is the old root HFS (version HFS). The recommendation is that it should be mounted read-only. Its mount point is created dynamically and the name of the HFS is the value specified on the VERSION statement in the BPXPRMxx member. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

7.78 Sysplex environment setup



- ❑ Create the Sysplex Root HFS
- ❑ Create the system-specific HFS
- ❑ Version HFS supplied by ServerPac
- ❑ Create OMVS couple data set (CDS)
- ❑ Define OMVS CDS to XCF
- ❑ Additional recommendations
  - ➢ Use standard naming convention for HFS data sets
  - ➢ Do not use &SYSNAME as a Root HFS qualifier
- ❑ Update BPXPRMxx member

Figure 7-78 Steps required to set up the shared sysplex environment

Figure 7-78 shows the steps involved in defining the HFS data sets for sharing in read/write mode in a sysplex environment. Some details are included in the following sections.

Sysplex root

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read-write and designated AUTOMOVE. Only one sysplex root is allowed for all systems participating in shared HFS. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB.

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

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible.

Version HFS

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

System-specific HFS

Directories in the system-specific HFS data set are used as mount points, specifically for /etc, /var, /tmp, and /dev. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in shared HFS). IBM recommends that the name of the system-specific data set contain the system name as one of the qualifiers. This allows you to use the &SYSNAME. symbolic (defined in IEASYMxx) in BPXPRMxx.

OMVS couple data set

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

Define CDS to XCF

Following is the sample JCL with comments to define the CDS to XCF.

```
//STEP10    EXEC PGM=IXCL1DSU
//STEPLIB   DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
 /* Begin definition for OMVS couple data set(1)            */
    DEFINEDS SYSPLEX(PLEX1)
 /* Name of the sysplex in which the OMVS couple data set is to be used
     DSN(SYS1.OMVS.CDS01) VOLSER(3390x1)
 /* The name and volume for the OMVS couple data set.
 The utility will allocate a new data set by the name specified on the
 volume specified.*/
 MAXSYSTEMS(8)
 /* Specifies the number of systems to be supported by the OMVS CDS.
    Default =     8          */
    NOCATALOG
 /* Default is not to CATALOG */
 DATA TYPE(BPXMCDS)
  /* The type of data in the data set being created for OMVS.
BPXMCDS is the TYPE for OMVS. */
 ITEM NAME(MOUNTS) NUMBER(500)
 /* Specifies the number of MOUNTS that can be supported by OMVS.*/
    Default =   100
    Suggested minimum =    10
    Suggested maximum = 35000  */
 ITEM NAME(AMTRULES) NUMBER(50)
  /* Specifies the number of automount rules that can be supported by OMV
     Default =    50
     Minimum =    50
     Maximum =  1000           */
```

BPXPRMxx member

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

7.79 HFS data sets: Shared sysplex

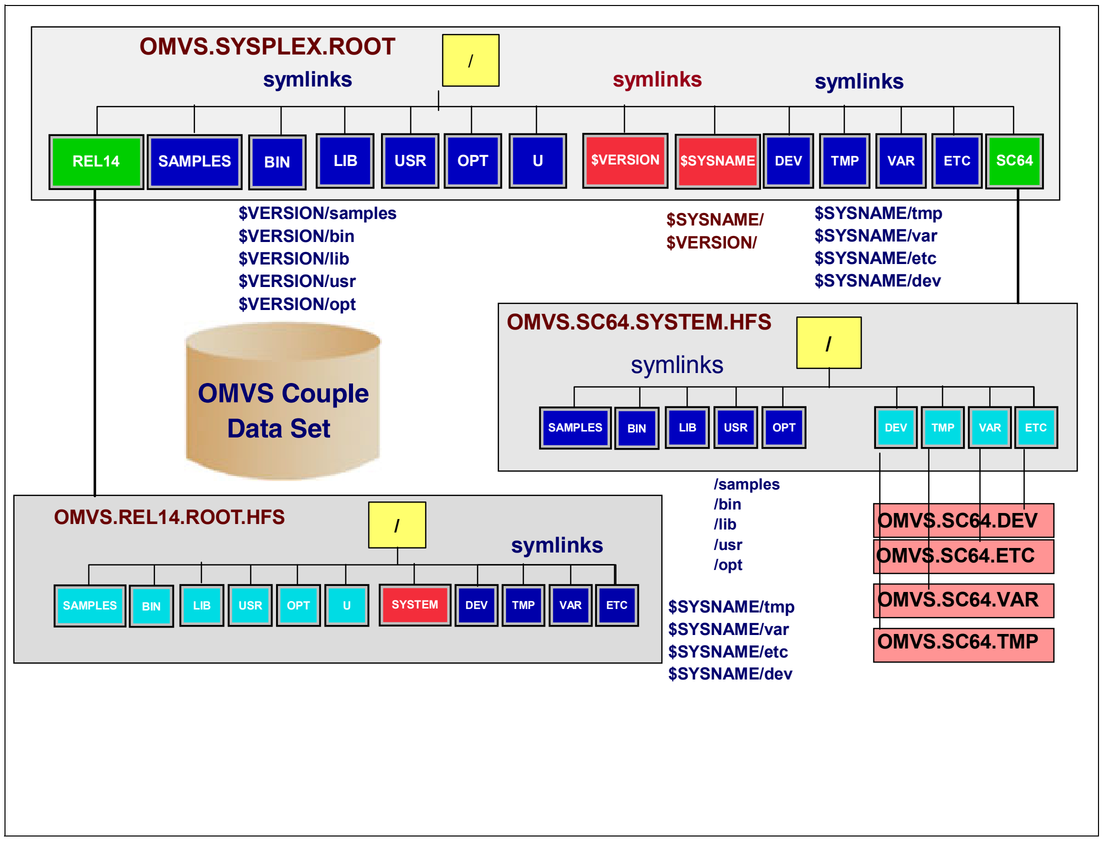


Figure 7-79 Shared sysplex HFS data sets

Sysplex root

No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set. The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible. The presence of symbolic links is transparent to the user.

System-specific HFS

The system-specific HFS data set should be mounted read-write, and should be designated NOAUTOMOVE. /etc, /var, /tmp, and /dev should also be mounted NOAUTOMOVE. In addition, IBM recommends that the name of the system-specific data set contains the system name as one of the qualifiers. This allows you to use the &SYSNAME. symbolic (defined in IEASYMxx) in BPXPRMxx.

Version HFS

The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, “root HFS” has been renamed to “version HFS.” IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one version HFS for each set of systems participating in shared HFS and that are at the same release level (that is, using the same SYSRES volume).

7.80 Multiple systems: Different versions

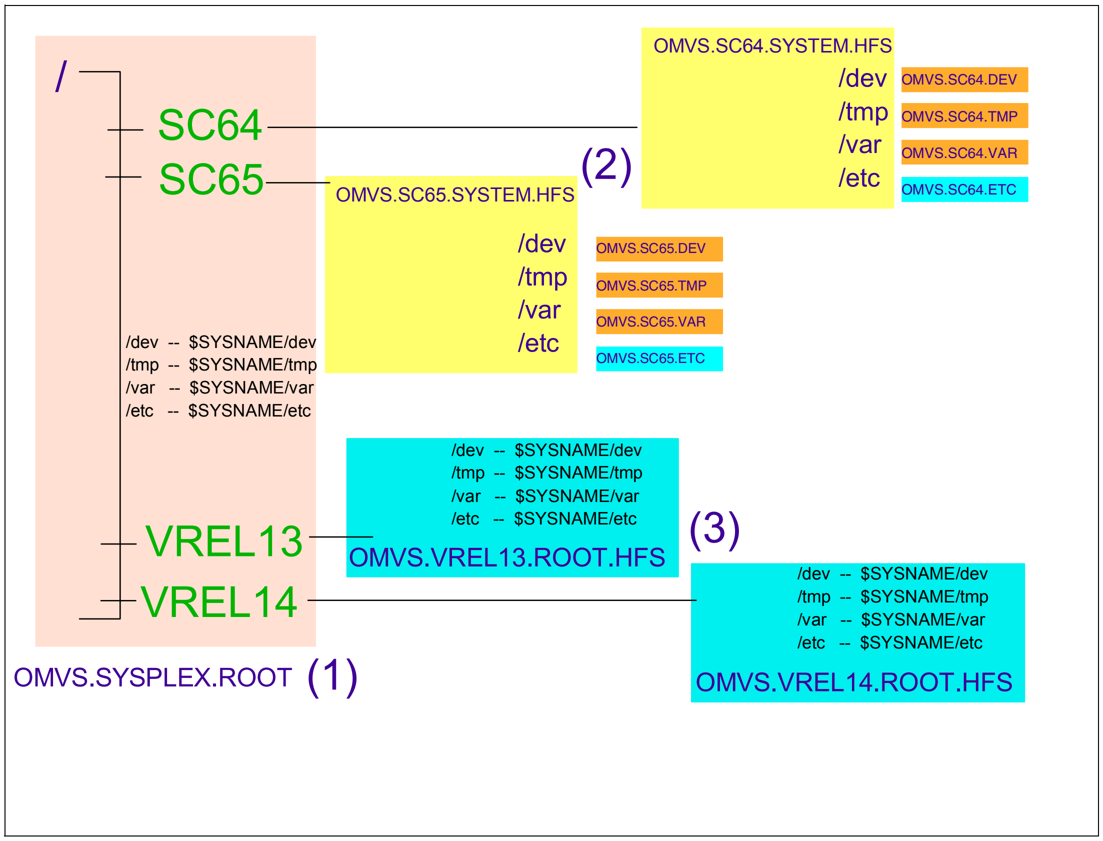


Figure 7-80 Shared sysplex with different z/OS levels

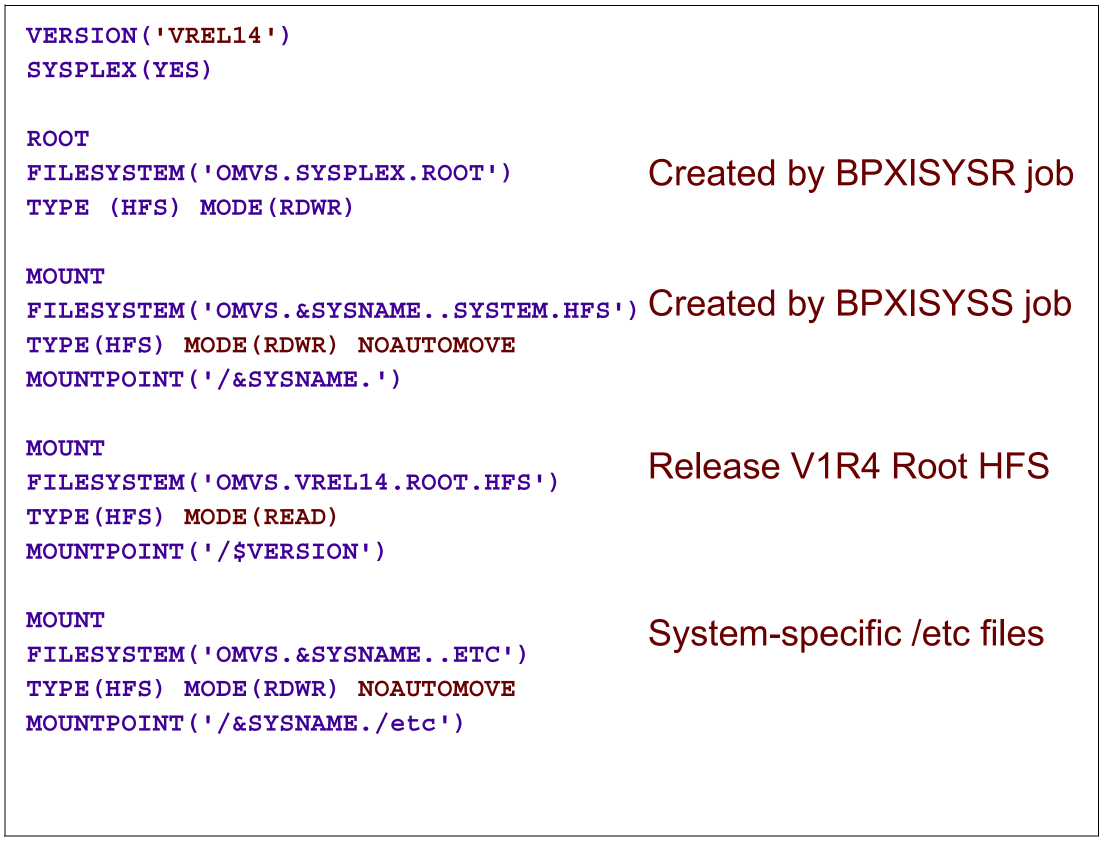
The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, “root HFS” has been renamed to “version HFS.”

IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one version HFS for each set of systems participating in shared HFS and that are at the same release level (that is, using the same SYSRES volume).

In Figure 7-80, there are two systems in the sysplex, SC54 and SC65. There are also two different z/OS releases, one on each system for z/OS V1R3 and z/OS V1R4. Therefore, there are two version roots mounted off of the sysplex root. The directories in the sysplex root are defined in the BPXPRMxx member for each system with the VERSION= statement, as follows:

```
VERSION('VREL13')          VERSION(‘VREL14)
SYSPLEX(YES)               SYSPLEX(YES)
```

7.81 Update BPXPRMxx for sysplex



```
VERSION('VREL14')
SYSPLEX(YES)

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

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

MOUNT
FILESYSTEM('OMVS.VREL14.ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

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

Created by BPXISYSR job
Created by BPXISYSS job
Release V1R4 Root HFS
System-specific /etc files

Figure 7-81 BPXPRMxx definitions for sysplex sharing

Sysplex sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbolics, as shown in Figure 7-81. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. The following parameters set up HFS sharing in a sysplex:

- `SYSPLEX(YES)` sets up sysplex sharing for those who are participating in HFS or zFS data sharing. To participate in HFS data sharing, the systems must be at the OS/390 V2R9 level or later. Those systems that specify `SYSPLEX(YES)` make up the participating group for the sysplex.
- `VERSION('`*`nnnn`*`')` allows multiple releases and service levels of the binaries to coexist and participate in HFS sharing. `nnnn` is a qualifier to represent a level of the version HFS. The most appropriate values for `nnnn` are the name of the target zone, &SYSR1., or another qualifier meaningful to the system programmer. A directory with the value nnnn specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version HFS.

BPXISYSR and BPXISYSS are provided as sample jobs to build root and system-specific HFS in SYS1.SAMPLIB.

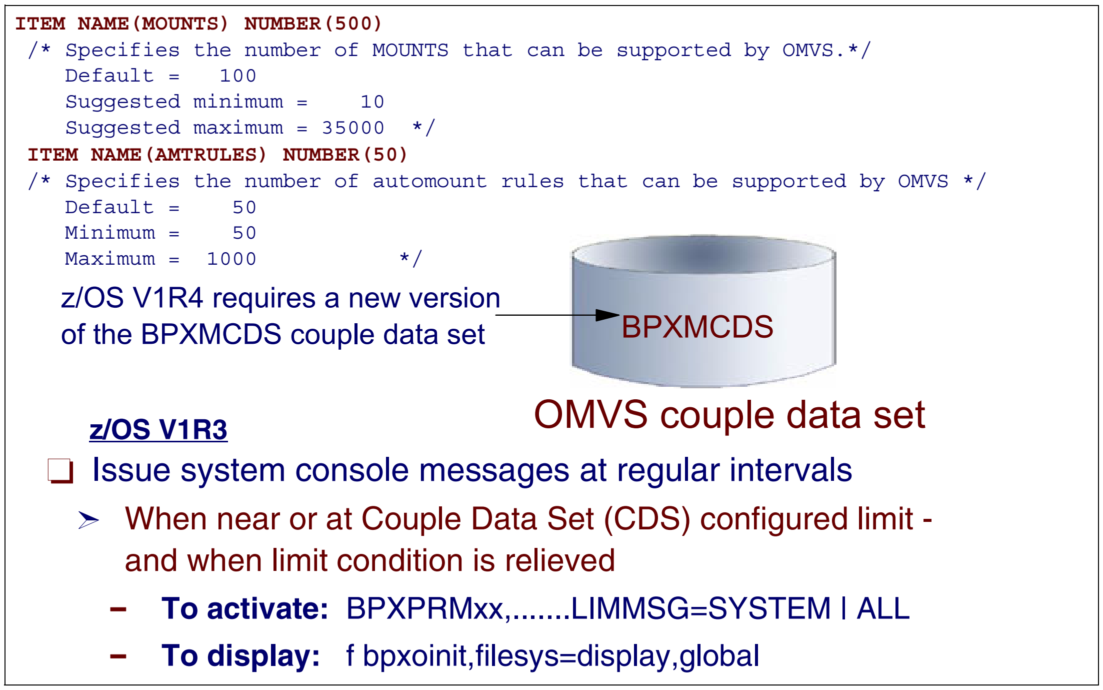
Jobs BPXISYS1 and BPXISYS2 are also provided to build root and system-specific HFS directories and symlinks.

The `AUTOMOVE|NOAUTOMOVE|UNMOUNT` parameters on **`ROOT`** and **`MOUNT`** indicate what happens to the file system if the system that owns that file system goes down, as follows:

- `AUTOMOVE` specifies that ownership of the file system is automatically moved to another system. It is the default.
- `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.

You should define your version and sysplex root HFS 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 defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted.

7.82 OMVS couple data set



```
ITEM NAME(MOUNTS) NUMBER(500)
 /* 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 =  1000           */
```

z/OS V1R4 requires a new version of the BPXMCDS couple data set

BPXMCDS

OMVS couple data set

z/OS V1R3

- Issue system console messages at regular intervals
  - When near or at Couple Data Set (CDS) configured limit - and when limit condition is relieved
    - **To activate:** BPXPRMxx,.......LIMMSG=SYSTEM | ALL
    - **To display:** f bpxoinit,filesys=display,global

Figure 7-82 Defining the OMVS couple data set

Shared sysplex support uses a type BPXMCDS couple data set (CDS) to maintain data about mounted file systems in the sysplex configuration. The primary and alternate CDSs are formatted, using the IXCL1DSU utility, with a maximum number of mount entries as specified in the `NUMBER` value that specifies the number of mounts, as shown in Figure 7-82.

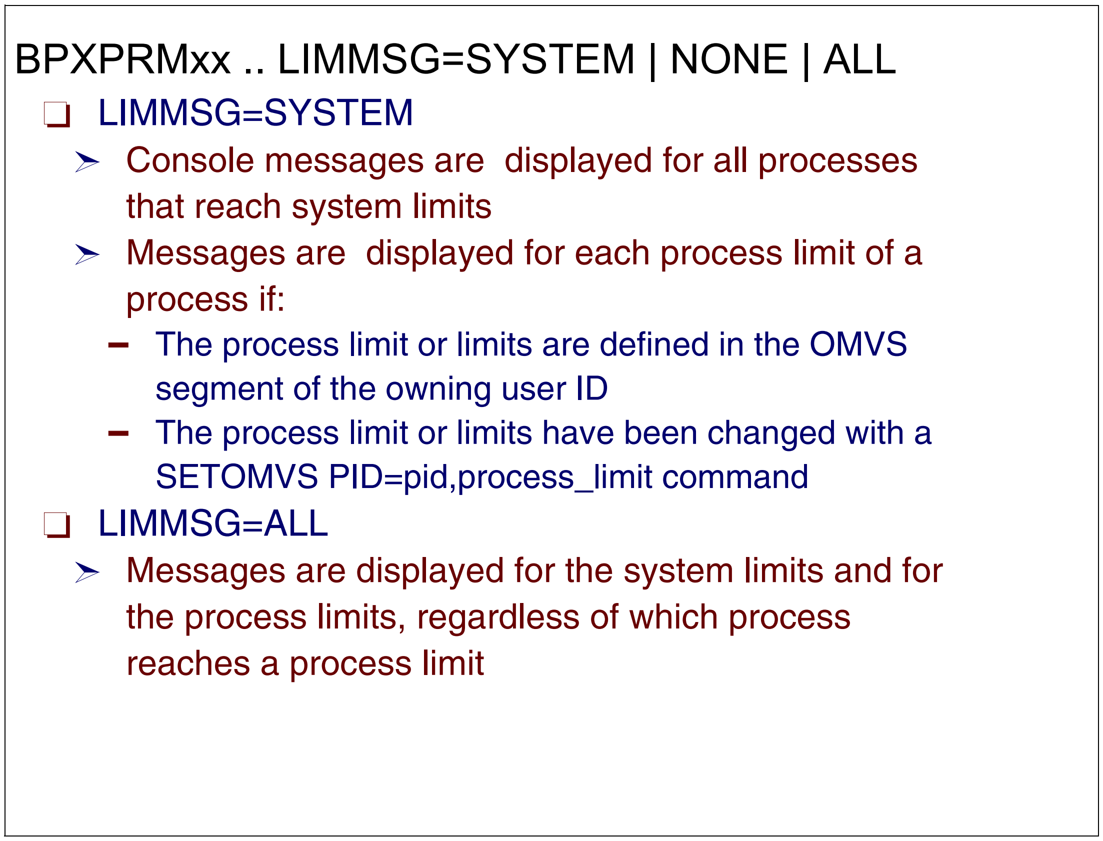
In previous releases, users needed the capability to determine when the number of file system mounts in a shared sysplex was approaching the configured limit. Before z/OS V1R3 there was no way to easily determine when the mount limit specified in the BPXMCDS CDS was being approached. z/OS V1R3 introduces the possibility of monitoring the shared HFS mount limits, specified in the CDS, by issuing a console message when the limit has almost been reached.

Once the mount limit is reached, no more file systems can be mounted in the sysplex until a larger type BPXMCDS CDS is enabled. Mount table limit monitoring allows an installation to detect when a primary CDS is reaching its mount table limit in order to begin corrective actions before denial of service.

You can display the number of mount entries and the number in use by issuing the **`F BPXOINIT,FILESYS=DISPLAY,GLOBAL`** command:

```
f bpxoinit,filesys=display,global
BPXF041I 2002/05/09 13.42.25 MODIFY BPXOINIT,FILESYS=DISPLAY,GLOBAL
221
SYSTEM   LFS VERSION ---STATUS--------------------- RECOMMENDED ACTION
SC64       1.  3.  1 VERIFIED                       NONE
SC63       1.  3.  1 VERIFIED                       NONE
SC65       1.  4.  1 VERIFIED                       NONE
CDS VERSION=  1        MIN LFS VERSION=  1.  3.  1
BRLM SERVER=N/A        DEVICE NUMBER OF LAST MOUNT=     706
MAXIMUM MOUNT ENTRIES=     500   MOUNT ENTRIES IN USE=     430
```

7.83 Defining process limits



BPXPRMxx .. LIMMSG=SYSTEM | NONE | ALL

- LIMMSG=SYSTEM
  - Console messages are displayed for all processes that reach system limits
  - Messages are displayed for each process limit of a process if:
    - The process limit or limits are defined in the OMVS segment of the owning user ID
    - The process limit or limits have been changed with a SETOMVS PID=pid,process_limit command
- LIMMSG=ALL
  - Messages are displayed for the system limits and for the process limits, regardless of which process reaches a process limit

Figure 7-83 Defining limits for BPXPRMxx parameters

The LIMMSG values were defined in z/OS V2R10, and have the following meanings:

SYSTEM Console messages are to be displayed for all processes that reach system limits. In addition, messages are to be displayed for each process limit of a process if:

- The process limit or limits are defined in the OMVS segment of the owning user ID
- The process limit or limits have been changed with a **`SETOMVS PID=pid,proces_limit`** command

ALL Console messages are to be displayed for the system limits and for the process limits, regardless of which process reaches a process limit.

Beginning in z/OS V1R3, several massages are issues for mount table limit monitoring. As the capacity of the mount table is approached, the following message is displayed:

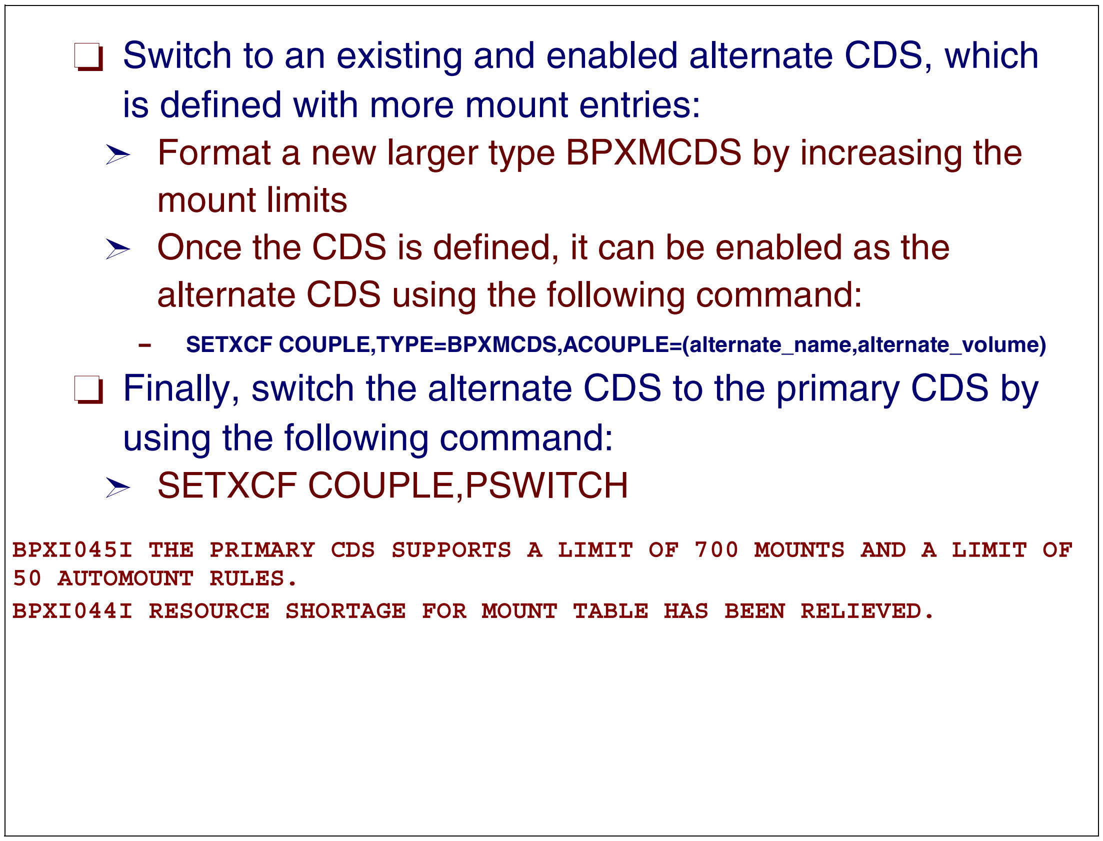
```
BPXI043E MOUNT TABLE LIMIT HAS REACHED <nn>% OF ITS CURRENT CAPACITY OF <current limit>.
```

where <nn> has the value of 85, 90, 95 or 100; the message is updated when the percentage has changed.

The message is DOM'd when the percentage decreases below 85%, and the following message is issued:

```
BPXI044I RESOURCE SHORTAGE FOR MOUNT TABLE HAS BEEN RELIEVED.
```

7.84 Mount limiting corrective action



- ❑ Switch to an existing and enabled alternate CDS, which is defined with more mount entries:
  - ➢ Format a new larger type BPXMCDS by increasing the mount limits
  - ➢ Once the CDS is defined, it can be enabled as the alternate CDS using the following command:
    - – SETXCF COUPLE,TYPE=BPXMCDS,ACOUPLE=(alternate_name,alternate_volume)
- ❑ Finally, switch the alternate CDS to the primary CDS by using the following command:
  - ➢ SETXCF COUPLE,PSWITCH

```
BPXI045I THE PRIMARY CDS SUPPORTS A LIMIT OF 700 MOUNTS AND A LIMIT OF
50 AUTOMOUNT RULES.
BPXI044I RESOURCE SHORTAGE FOR MOUNT TABLE HAS BEEN RELIEVED.
```

Figure 7-84 Action to take when mount limit is reached in OMVS couple data set

Corrective action may consist of one of the following procedures:

- Format a new, larger type BPXMCDS. A sample job to create and format a type BPXMCDS can be found in SYS1.SAMPLIB(BPXISCDS). Once the CDS is defined, it can be enabled as the ALTERNATE CDS using the following system command:

  ```
  SETXCF COUPLE,TYPE=BPXMCDS,ACOUPLE=(alternate_name,alternate_volume)
  ```

- Switch the alternate CDS to the primary CDS by using the following system command:

  ```
  SETXCF COUPLE,PSWITCH
  ```

7.85 Mounting shared sysplex file systems

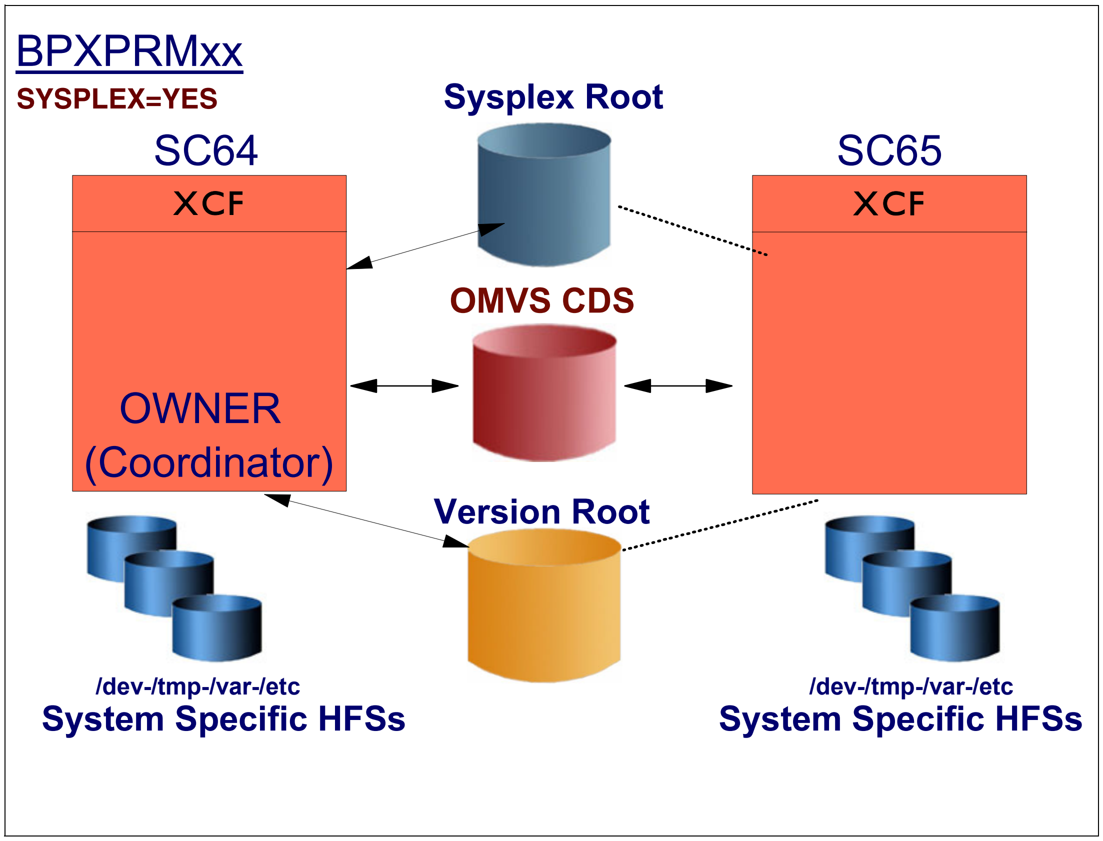


Figure 7-85 Shared sysplex mounted file systems

With a shared sysplex environment, sharing HFS and zFS file systems, each system must have specific HFS data sets for each of these file systems, meaning the two systems each have HFS data sets for /etc, /tmp, /var, and /dev. The file systems are then mounted under the system-specific HFS, as shown in Figure 7-85. With this shared HFS support, one system can access system-specific file systems on another system. (The existing security model remains the same.) For example, while logged onto SC65, you can gain read-write access to SC64's /tmp by specifying /SC64/tmp/.

You should also be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in shared HFS. The easiest way to accomplish this is to create a single BPXPRMxx member that contains file system information for each system participating in shared HFS. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible.

Note: The first system that IPLs in the sysplex becomes the owner of the sysplex root.

7.86 Accessing shared sysplex file systems

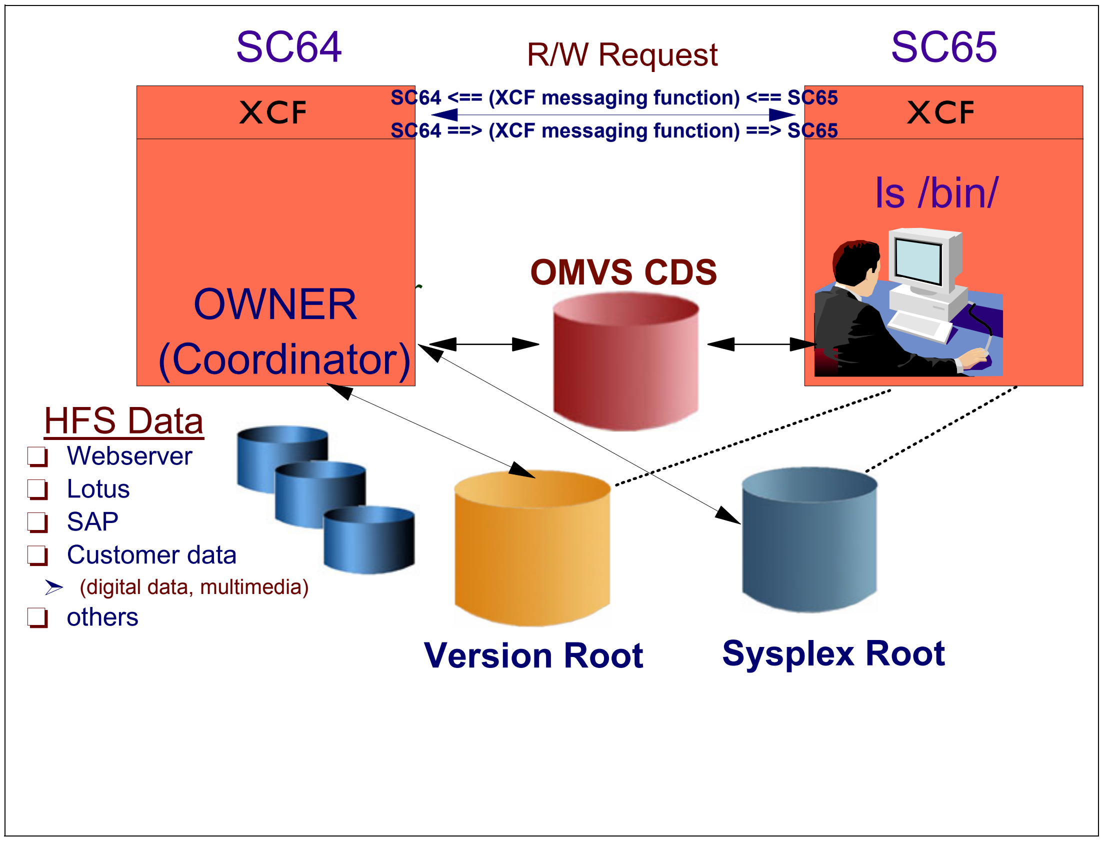


Figure 7-86 Accessing shared sysplex mounted file systems

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

For example, assume that a user on SC64 requests a read on a file system mounted R/W and owned by SC65. Using shared HFS support, SC64 sends a message requesting this read to SC65 via an XCF messaging function:

SC64 ===> (XCF messaging function) ===> SC65

After SC65 gets this message, it issues the read on behalf of SC64, and gathers the data from the file. It then returns the data via the same route the request message took:

SC65 ===> (XCF messaging function) ===> SC64

7.87 Shared file system AUTOMOVE takeover

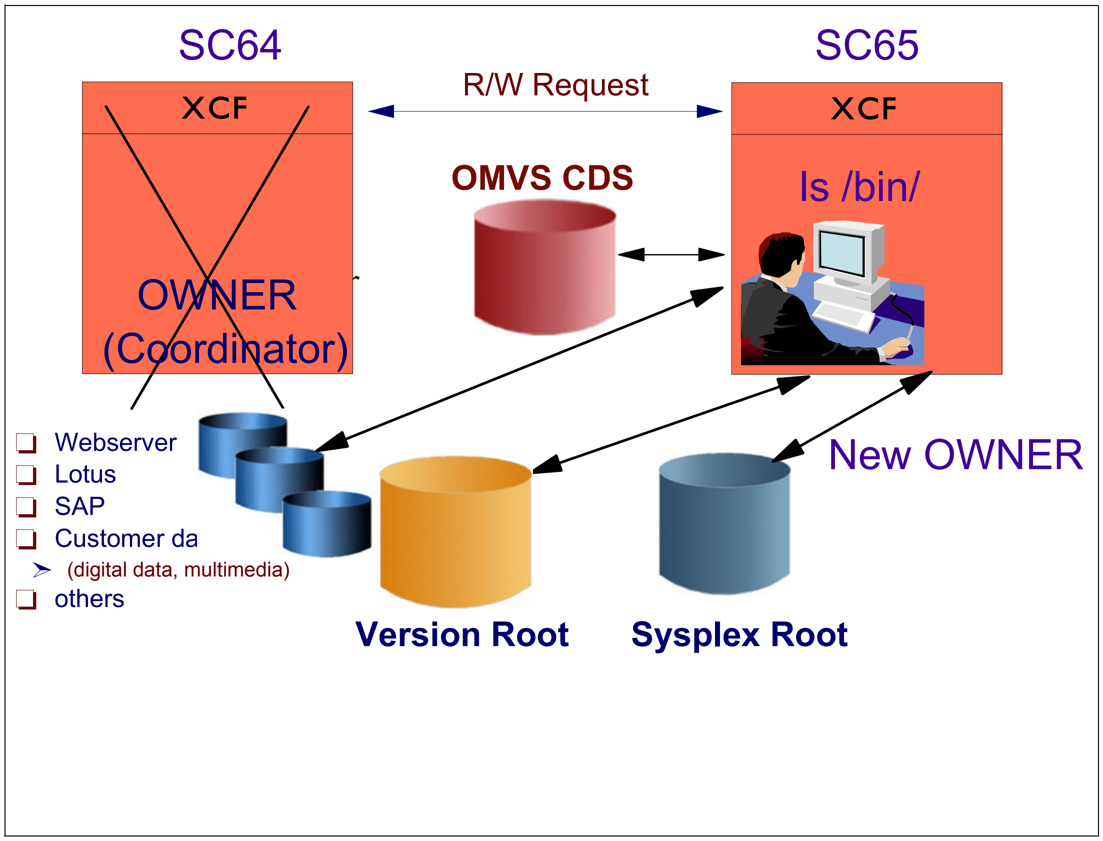


Figure 7-87 AUTOMOVE options in a shared sysplex

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

Generally, when an owning system fails, ownership over its automove-mounted file system is moved to another system and the file is usable. However, if a file system is mounted read-write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered.

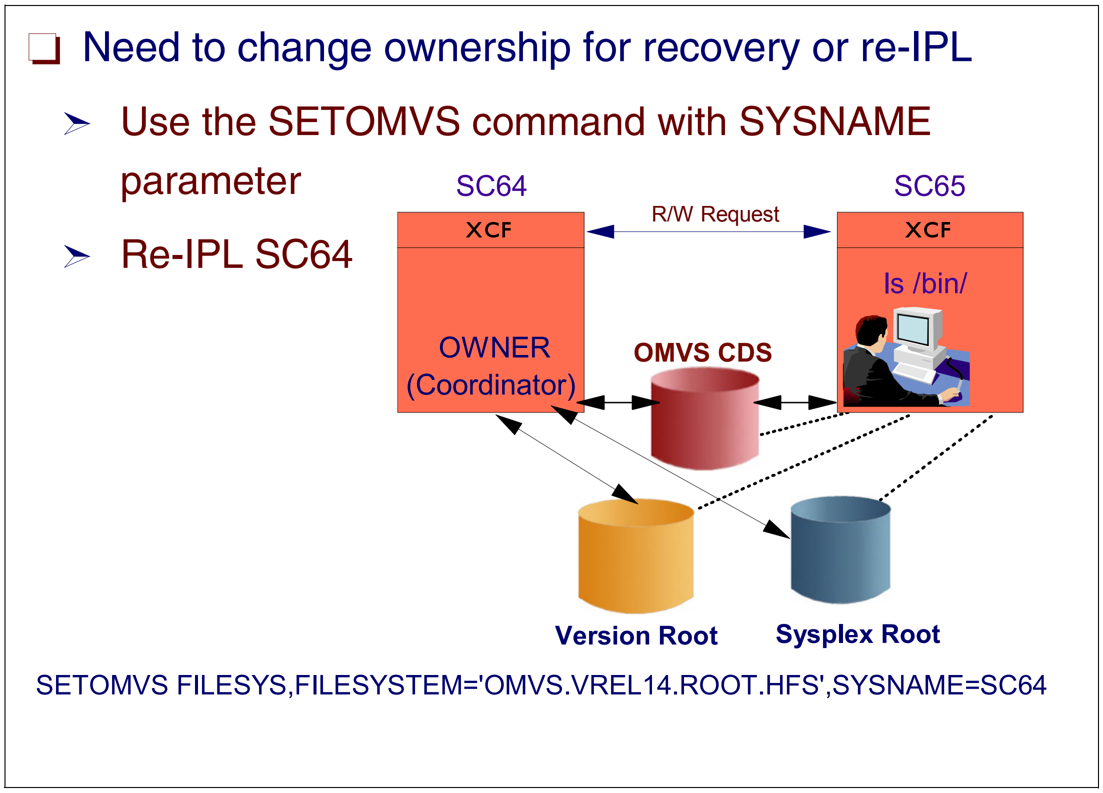
For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

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

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

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

7.88 Moving file systems in a sysplex



❑ Need to change ownership for recovery or re-IPL

- Use the SETOMVS command with SYSNAME parameter
- Re-IPL SC64

SETOMVS FILESYS,FILESYSTEM='OMVS.VREL14.ROOT.HFS',SYSNAME=SC64

Figure 7-88 Command to move a file system in a sysplex environment

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

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

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

Examples of moving file systems are:

- To move ownership of the file system that contains /u/user1 to SC64:

```
chmount -d SC64 /u/user1
```

- To move ownership of the payroll file system from the current owner to SC64 using SETOMVS, issue:

```
SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SC64
```

- Assuming the mount point is over directory /PAYROLL:

```
SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SC64
```

- Move the root file system:

```
SETOMVS FILESYS,FILESYSTEM='OMVS.VREL14.ROOT.HFS',SYSNAME=SC64
```

7.89 Non-volatile root file system

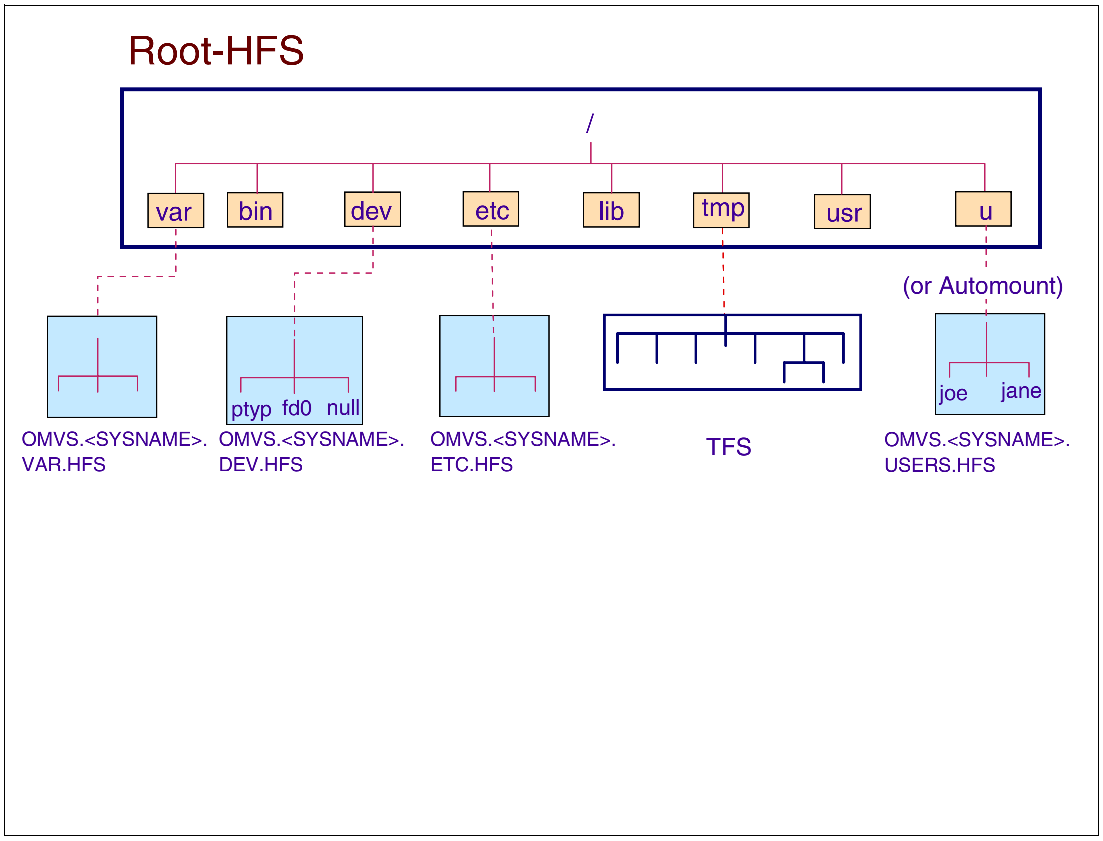


Figure 7-89 Managing the root file system

Extra planning should be done to keep file activity out of the root file system. This will protect the root and make future migration to new system levels easier.

Installation customized files (such as profiles in /etc) should be kept in their own HFS. This will protect these files from being overlaid by a future system replace.

Only file systems that are mounted in read-only mode can be shared between multiple system images. However, IBM no longer recommends mounting the root file system in read-only mode since it will cause customization problems and incorrect setup.

z/OS is built on a *system replace* basis. As new levels come out, all systems data sets, including the root HFS, will be replaced. Therefore, make a plan to get all of your tailored files out of the root into their own HFS. Then when the system is replaced, you will only have to do minimal revising of your profiles and other user-oriented files and remount their HFSs to the new root. Keeping changes out of the root will make future changes much easier to manage.

7.90 Installation of other products

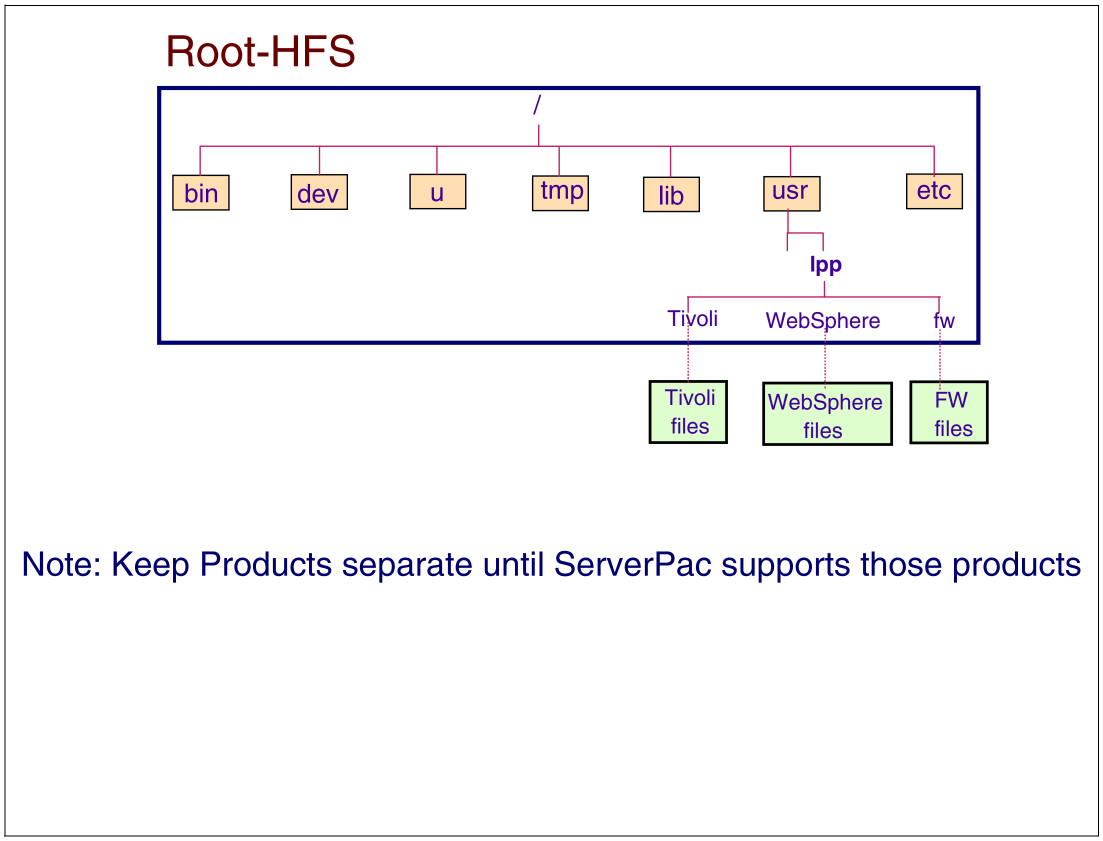


Figure 7-90 Installing products that are not part of the ServerPac

When installing products that are not part of the ServerPac order and that install into the HFS, consider the following:

- Create new directories where the files associated with the new products will be installed.
- If possible, create a new HFS data set and mount it to the new directory. After installation of the product, all the files will reside in the new HFS data set.
- IBM has created the /usr/lpp directory and each product that puts files in the HFS structure creates a directory in lpp and then creates and mounts its own HFS at that mount point.
- Keeping new products in different HFS data sets offers better file system management while maintaining a more stable root file system. This also ensures easier maintenance when applying service.

Figure 7-90 shows the IBM convention for managing their products. Directory /usr/lpp is built during the build of the root. As products (such as DCE or ICS) are built, they will create a directory in /usr/lpp and then will mount an HFS to that directory. All the product files will then be installed in a separate file system. Since the main installation vehicle for z/OS is system replace, all these directories are built in root as it is built. System replace should include these products.

7.91 File system failure

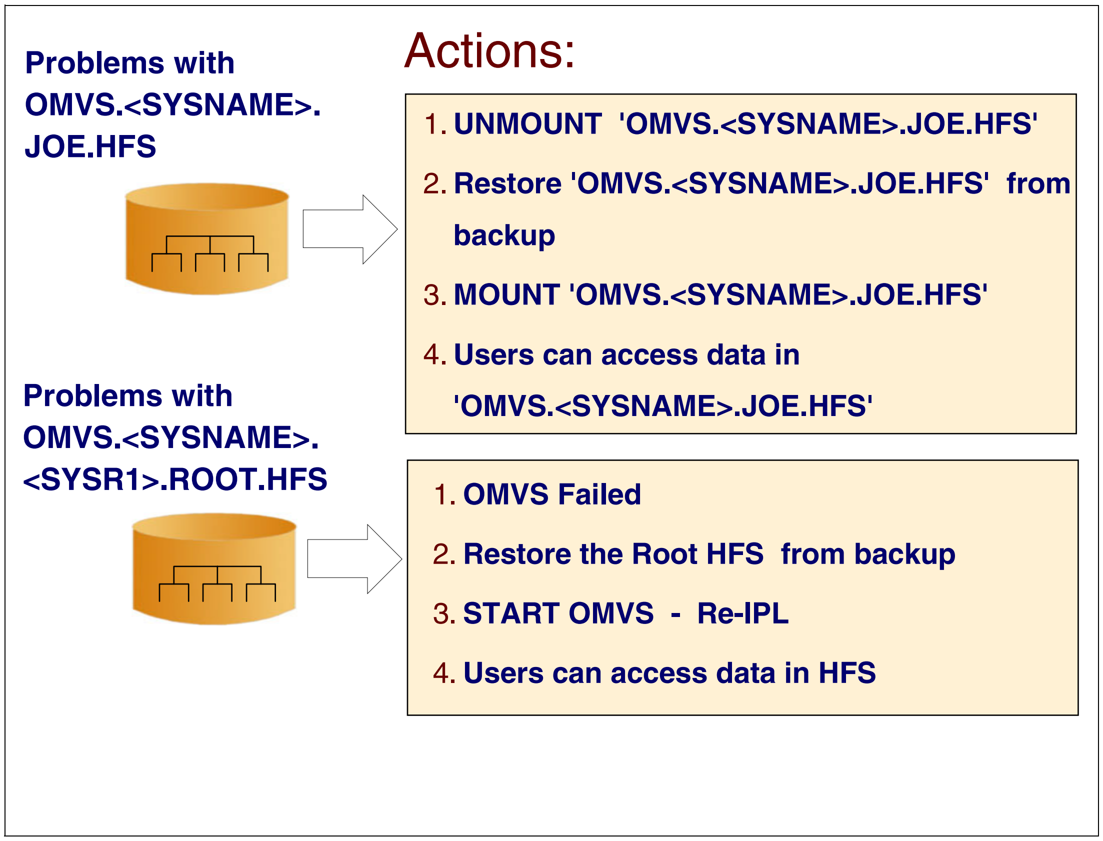

Problems with OMVS.<SYSNAME>. JOE.HFS
Actions:
1. UNMOUNT 'OMVS.<SYSNAME>.JOE.HFS'
2. Restore 'OMVS.<SYSNAME>.JOE.HFS' from backup
3. MOUNT 'OMVS.<SYSNAME>.JOE.HFS'
4. Users can access data in 'OMVS.<SYSNAME>.JOE.HFS'
Problems with OMVS.<SYSNAME>. <SYSR1>.ROOT.HFS
1. OMVS Failed
2. Restore the Root HFS from backup
3. START OMVS - Re-IPL
4. Users can access data in HFS

Figure 7-91 Handling HFS file system failures

If the system detects a failure in the hierarchical file system, the following message is issued:

```
BPXF014D FILESYSTYPE type TERMINATED.  REPLY 'R' WHEN READY TO RESTART
```

If there is a problem with a user file system, a system administrator or an operator can do the following:

1. Unmount the file system in error if this was not done by the system.
2. Restore the HFS data set from a backup copy.
3. Mount the restored HFS data set. Superuser authority is needed.
4. Send a message to all z/OS UNIX users affected by the error in this file system, that a back-level copy is in use. (Use the `wall` command.)

If there is a problem with the root file system, the options to recover are similar:

1. If the root file system fails, z/OS UNIX will fail.
2. Restore the HFS data set from a backup copy.
3. Edit BPXPRMxx and re-IPL.

When a serious problem occurs in a file system, and the data set must be restored from a backup copy, all data entered since the backup is lost. DFSMShsm can be used to take backup and restore of HFS data sets if DFSMSdss™ is used as the data mover.

If the root file system fails, z/OS UNIX will stop immediately. UNIX System Services cannot be active without a root file system.

7.92 File system access

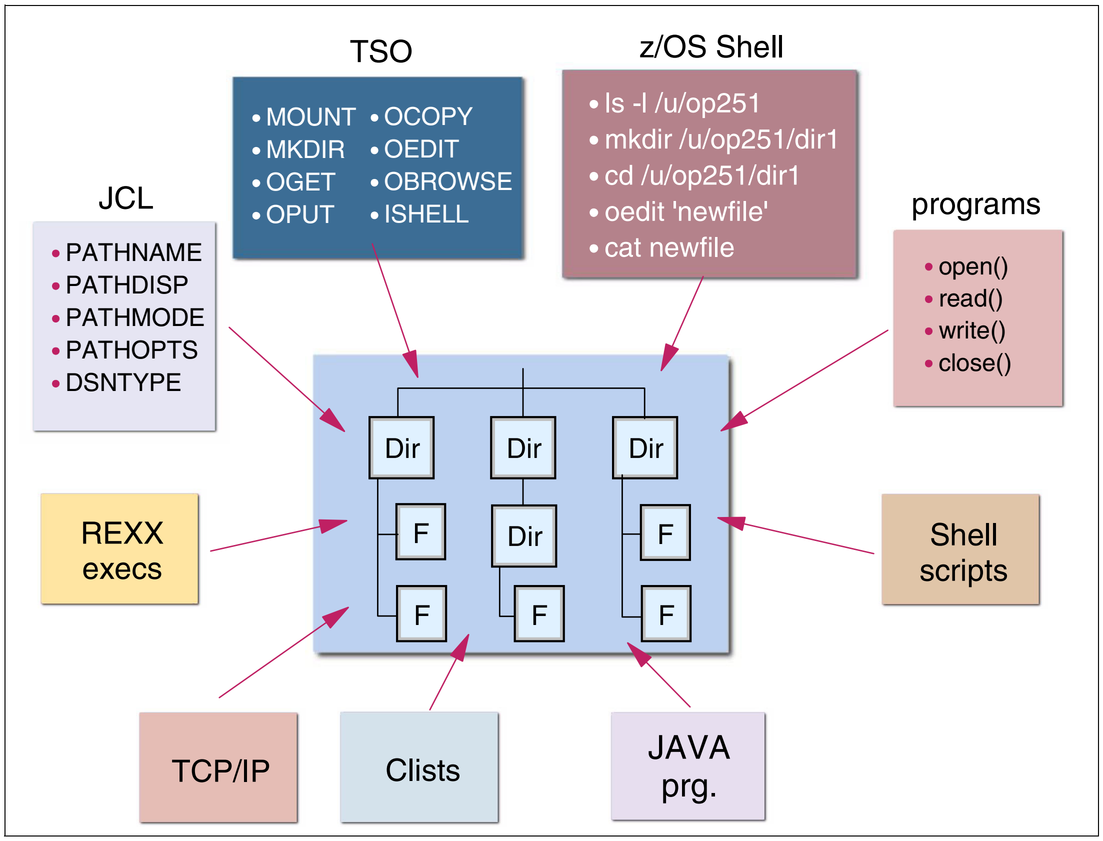


Figure 7-92 Accessing file system data from different methods

We have seen how we can customize the HFS for use. But who can use the HFS and the z/OS UNIX?

The z/OS UNIX file system can be used or accessed by using any of the following:

TSO/E	TSO/E has commands for browsing and editing HFS files, for copying data between HFS files and MVS data sets, to mount file systems, to invoke the z/OS UNIX shell. There is also a command called ISHELL which provides an ISPF menu-driven interface to the file system.
JCL	JCL provides keywords which support the specification of HFS pathnames.
REXX	REXX has a set of z/OS UNIX extensions (`syscall` commands) to access z/OS UNIX callable services.
z/OS UNIX shell	The shell is the UNIX interface to the file system. It contains commands and utilities to access HFS files.
C programs	z/OS UNIX supports many C functions to access HFS files.
Shell scripts	Shell scripts are similar to REXX execs. UNIX System Services shell commands and utilities can be stored in a text file which can be executed.

The HFS data set cannot be OPENed by any MVS utilities. Only DFSMSdss can be used to dump and restore this data set type. The internal structure of an HFS data set is only accessible via z/OS UNIX System Services.

7.93 File access

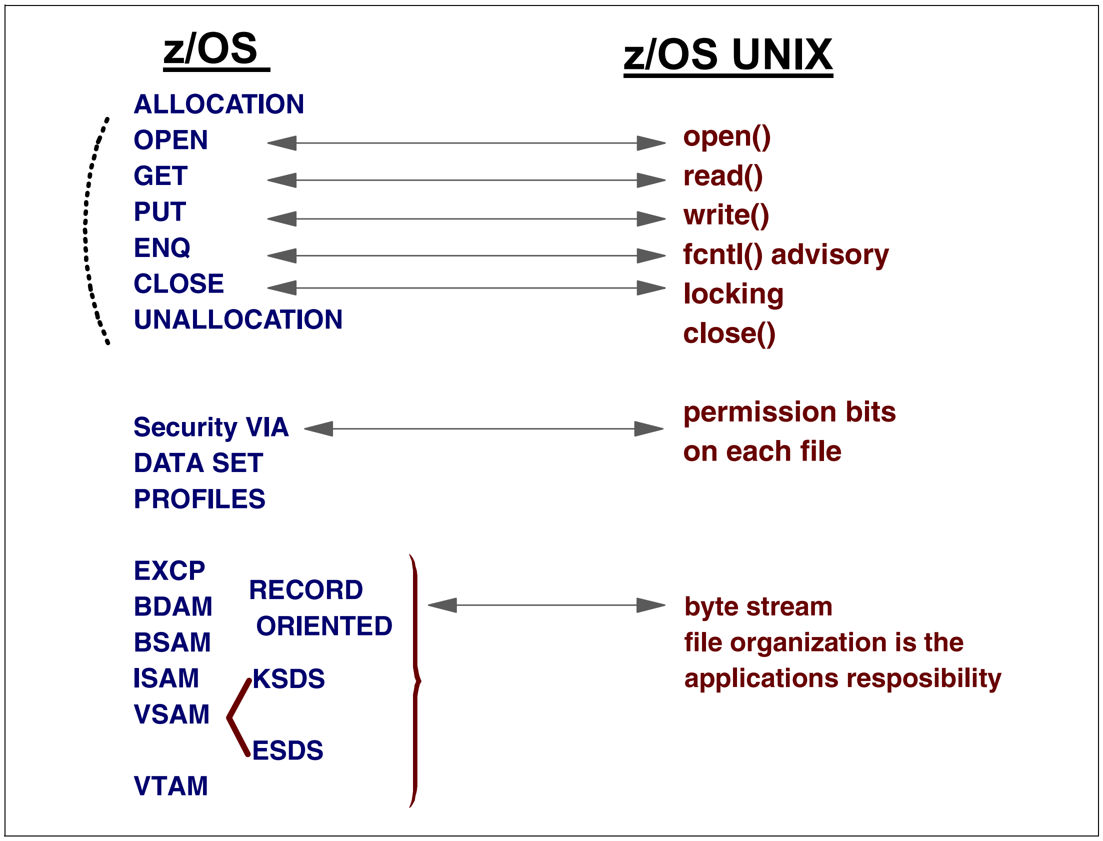
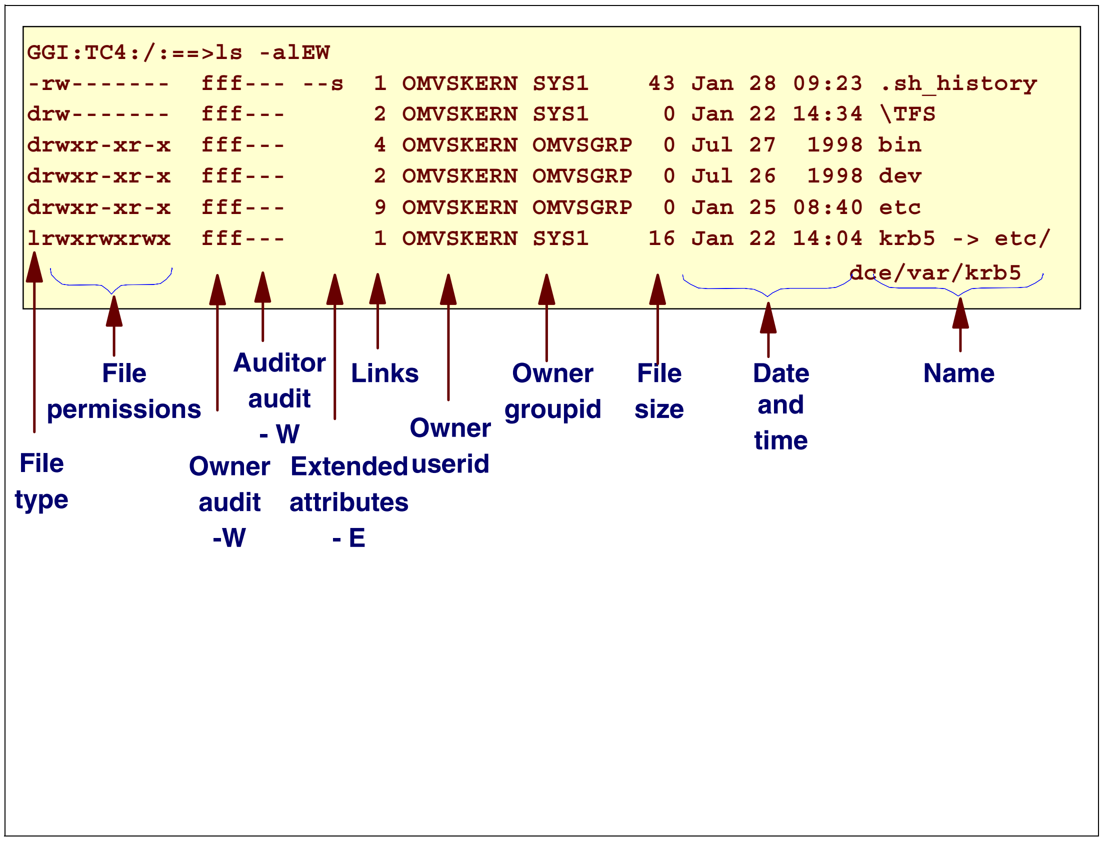


Figure 7-93 File system access versus MVS data set access

Figure 7-93 shows corresponding file access methods in z/OS and the POSIX standard.

The big difference is only the file handling. While OS/390 gives you file handle methods (you do not have to implement this), the POSIX standard does not define the file handling methods. So any application running on the POSIX standard is responsible for storing and retrieving data from and to the file.

7.94 List file and directory information



```
GGI:TC4:/:==>ls -alEW
-rw-------  fff--- --s  1 OMVSKERN SYS1     43 Jan 28 09:23 .sh_history
drw-------  fff---      2 OMVSKERN SYS1      0 Jan 22 14:34 \TFS
drwxr-xr-x  fff---      4 OMVSKERN OMVSGRP  0 Jul 27  1998 bin
drwxr-xr-x  fff---      2 OMVSKERN OMVSGRP  0 Jul 26  1998 dev
drwxr-xr-x  fff---      9 OMVSKERN OMVSGRP  0 Jan 25 08:40 etc
lrwxrwxrwx  fff---      1 OMVSKERN SYS1     16 Jan 22 14:04 krb5 -> etc/
                                                             dce/var/krb5
```

Figure 7-94 Command to access file and directory information

Figure 7-94 shows the output of the `ls -alEW` command.

- File type describes the type of file (for example, d Directory, l Symbol link, c Character special file, f Regular file, and so on).
- File permissions are Read Write Execute for user group and other. Sticky Bit.
- Owner audit (f for failed, s for successful access if audit attribute is set).
- Auditor audit is the same as for owner audit.
- External attributes.
- Links are the number of links to the file.
- Owner user ID shows which user ID owns the file or directory.
- Owner Group ID shows the name of the group that owns this file or directory.
- File size is the size of the file in bytes.
- Date and Time shows the date and time of change or creation.
- Name specifies a file name or link pointing to the file name.

7.95 File security packet - extattr bits

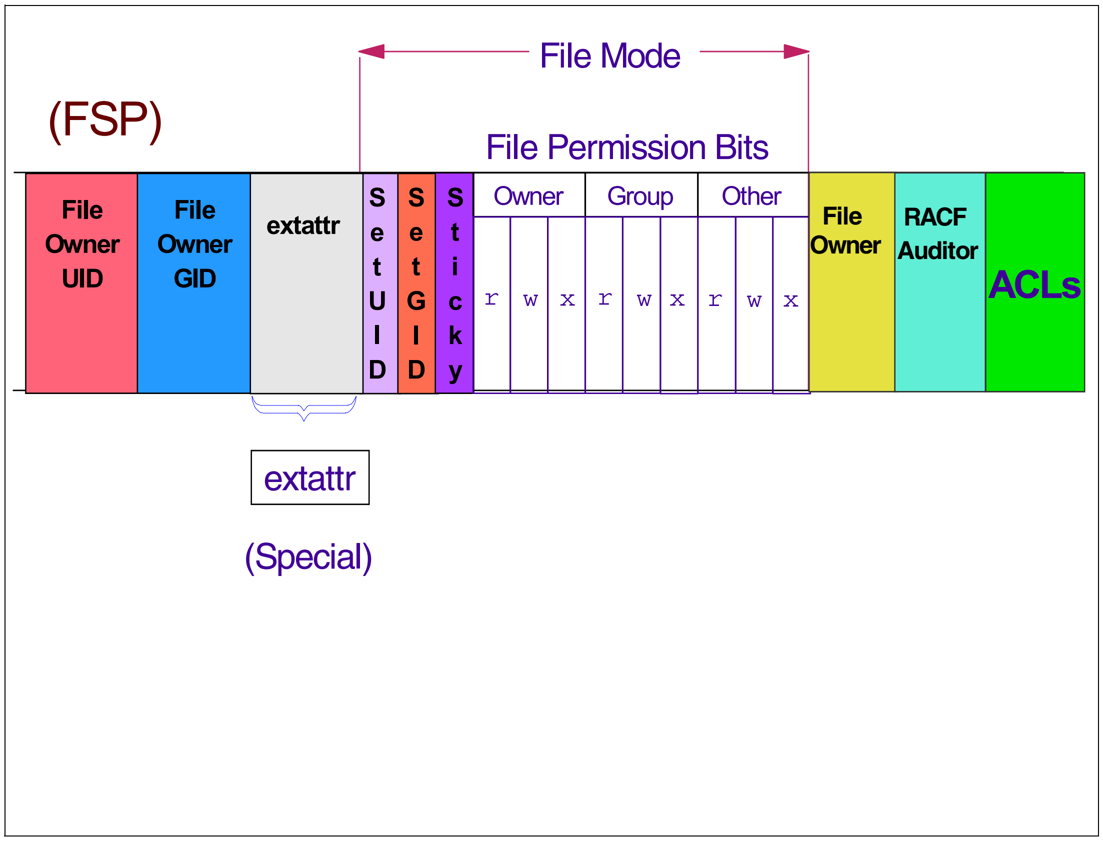


Figure 7-95 Extended attribute bits in the FSP

The extended attributes are kept in the file security packet (FSP). The extended attributes give special authorities to the files. The following sections describe their function and how to display them.

7.96 Extended attributes

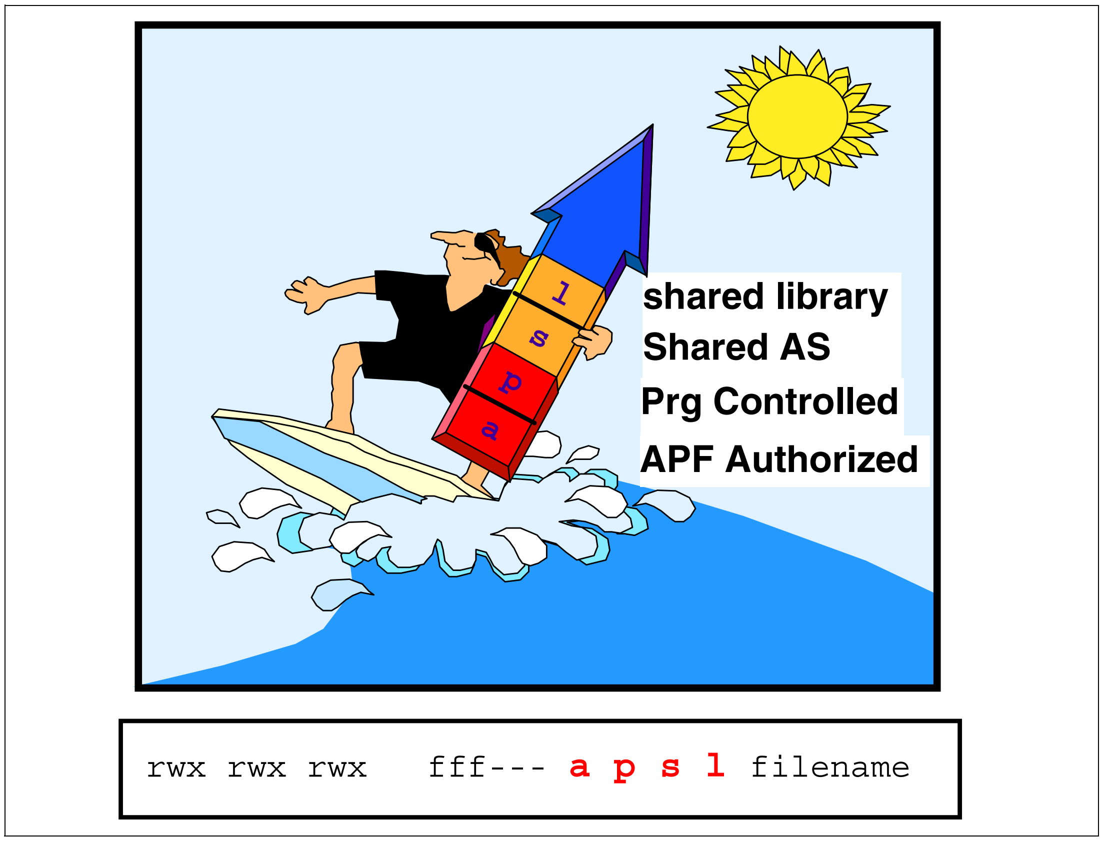


Figure 7-96 Listing the extended attributes

The extended attributes give special authorities to the files. Four different extended attributes are defined:

APF Authorized programs	The behavior of these programs is the same as other programs that are loaded from APF-authorized libraries.
Program-controlled program	All programs that are loaded into an address space that requires daemon authority need to be marked as "controlled."
Shared AS	The program shares its address space with other programs.
Shared library	Programs using shared libraries contain references to the library routines that are resolved by the loader at run time.

a	Program runs APF-authorized if linked AC=1
p	Program is considered program-controlled
s	Program runs in a shared address space
l	Program is loaded from the shared library region

To display or set the extended attribute bits, the following commands can be used:

- **`ls -E`**
- **`extattr`**

The status of the extended attributes can be shown with the following commands:

ls	Sticky Bit and Set UID/GID Bit
ls -E	APF Auth., Program Control, Shared AS, and shared library
l**s -W**	RACF Audit

With the following commands you can set the different extended attributes:

chmod	Sticky Bit, Set UID/GID
mkdir	Sticky Bit
extattr	Program Control, APF authorized, Shared AS

7.97 APF-authorized attribute

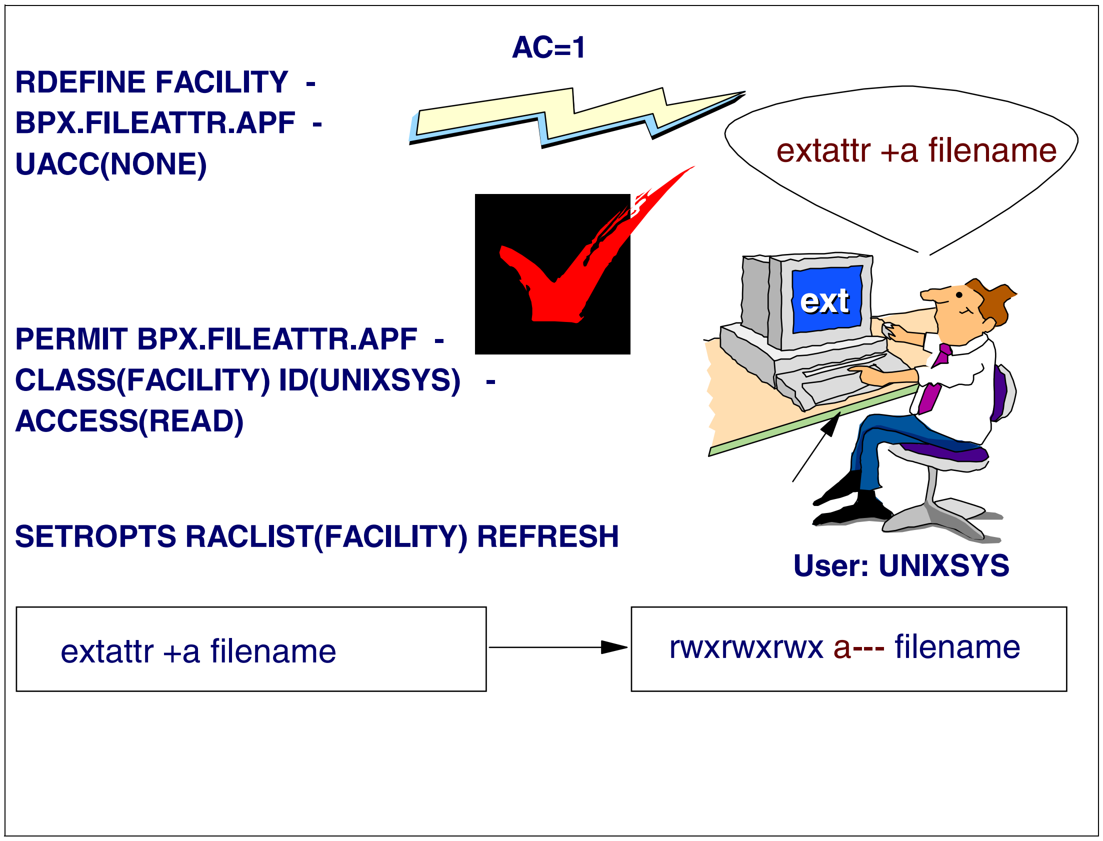


Figure 7-97 APF authorized attribute

The entire HFS is considered as an unauthorized library. You can authorize individual programs within the HFS as APF-authorized by setting the APF-extended attribute. If a program running in an APF-authorized address space attempts to load a program from the HFS 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.

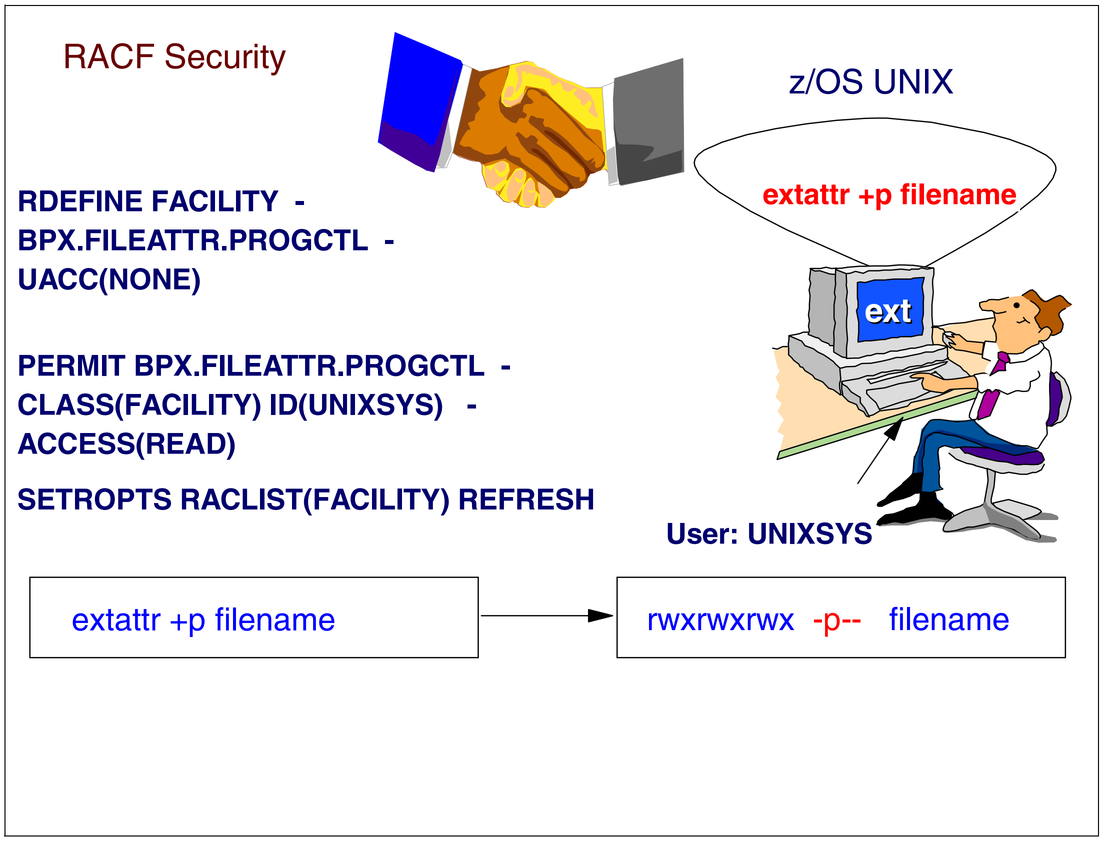
In order to activate APF authorization, the RACF profile BPX.FILEATTR.APF must be defined. The z/OS UNIX administrator must have read access to this profile to execute the **extattr** command, as follows:

- To turn on the APF-authorized program bit, issue:
```
extattr +a  filename
```
- To remove the APF-authorized program bit, issue:
```
extattr -a  filename
```

You can link-edit the program into an APF-authorized library and turn on the sticky bit in the HFS. You can use the **extattr** shell 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.

7.98 Activate program control



```
RDEFINE FACILITY -
BPX.FILEATTR.PROGCTL -
UACC(NONE)

PERMIT BPX.FILEATTR.PROGCTL -
CLASS(FACILITY) ID(UNIXSYS) -
ACCESS(READ)

SETROPTS RACLIST(FACILITY) REFRESH
```

Figure 7-98 Program control extended attribute

All programs loaded into an address space that requires daemon authority need to be marked as controlled. This means that user programs and any runtime library modules that are loaded must be marked as controlled.

You can mark programs in HFS files as controlled by turning on the extended attribute that allows you to run program-controlled. To turn this extended attribute on:

```
extattr +p  filename
extattr +p /user/sbin/proga
```

Only users with the correct permission can turn on the extended attribute. The example above shows the RACF command used to give this permission to z/OS UNIX administrator UNIXSYS. To turn off the extended attribute, use the **extattr** shell command:

```
extattr -p  filename
extattr -p /user/sbin/su
```

After a file is marked program-controlled, any activity that can change its contents results in the extended attribute being turned off. If this occurs, a system programmer with the appropriate privilege will have to verify that the file is still correct and reissue the **extattr** command to mark the file as program-controlled.

All modules loaded from LPA are considered to be controlled. RTLS libraries must be defined to RACF for program-controlled support.

7.99 Shared AS attribute

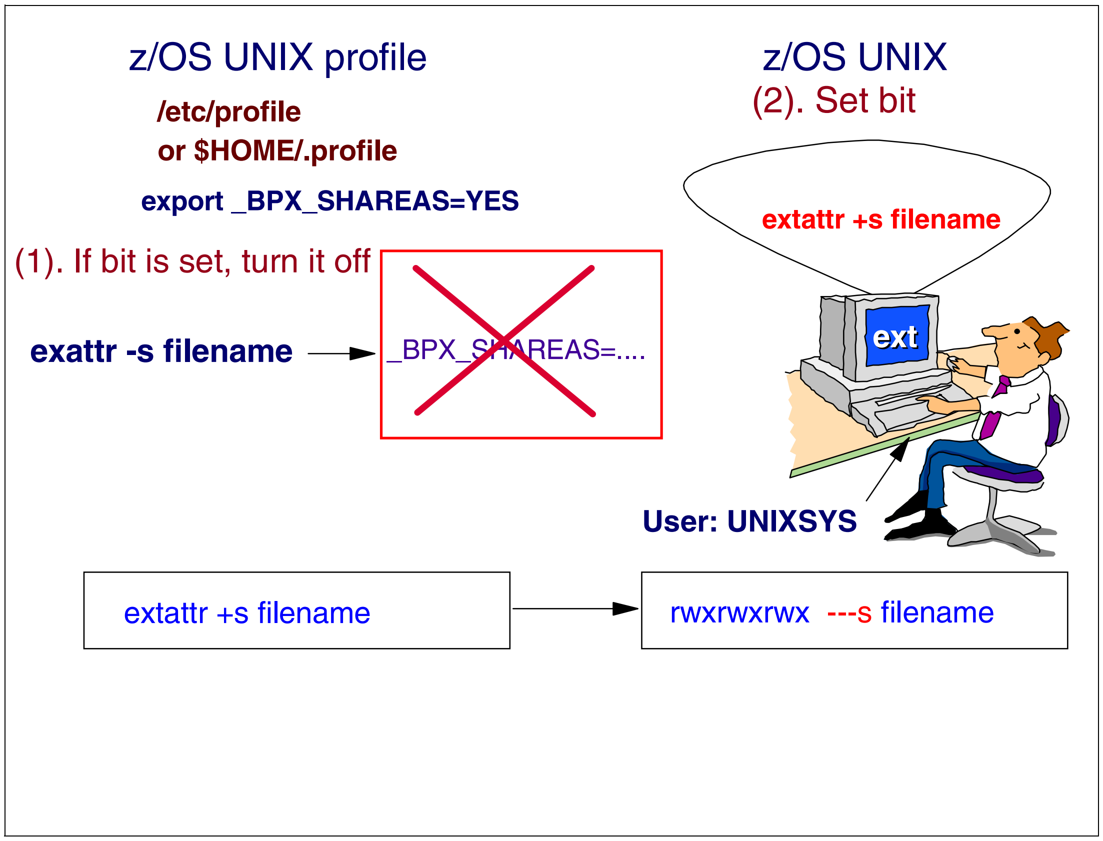


Figure 7-99 Shared AS extended attribute

To improve the performance in the shell, the extended attribute for shared AS can be set. The program will share this address space with other programs.

To turn this extended attribute on:

```
extattr +s  filename
```

To turn off the bit:

```
extattr -s  filename
```

When this attribute is not set (-s), the _BPX_SHAREAS environment variable is ignored when the file is spawn()ed. By default, this attribute is set (+s) for all executable files.

To improve performance for all shell users, it is recommended that /etc/profile or $HOME/.profile set the environment variable.

7.100 Shared library attribute

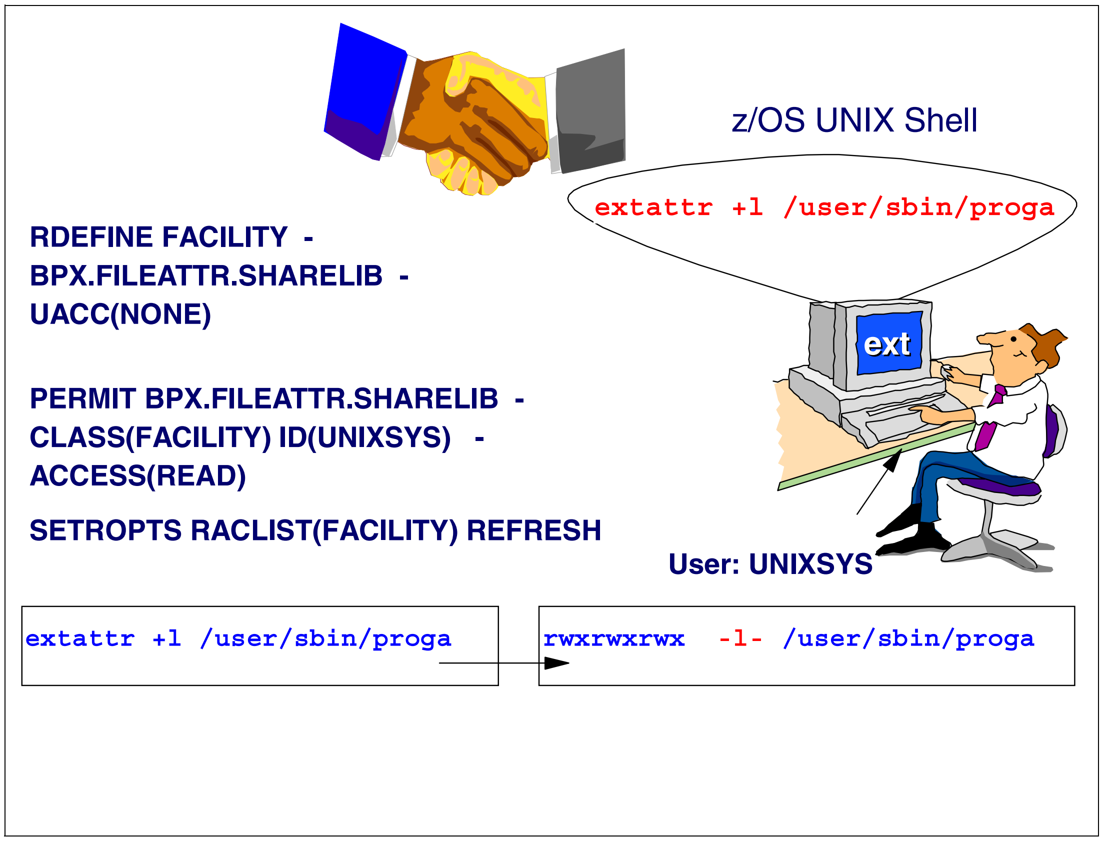

z/OS UNIX Shell
extattr +l /user/sbin/proga
RDEFINE FACILITY -
BPX.FILEATTR.SHARELIB -
UACC(NONE)
PERMIT BPX.FILEATTR.SHARELIB -
CLASS(FACILITY) ID(UNIXSYS) -
ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
ext
User: UNIXSYS
extattr +l /user/sbin/proga
rwxrwxrwx -l- /user/sbin/proga

Figure 7-100 Shared library extended attribute

When this attribute is set (+l) on an executable program file (load module), it will be loaded from the shared library region.

To be able to use the **extattr** command for the +l option, you must have at least READ access to the BPX.FILEATTR.SHARELIB FACILITY class. For more information, see *z/OS UNIX System Services Planning,* GA22-7800.

Note: l is a lower case L, not the numeral one or an upper case i.

The BPX.FILEATTR.SHARELIB FACILITY class profile controls who 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, proga is the name of the file.

```
extattr +l  /user/sbin/proga
```

7.101 Sticky bit

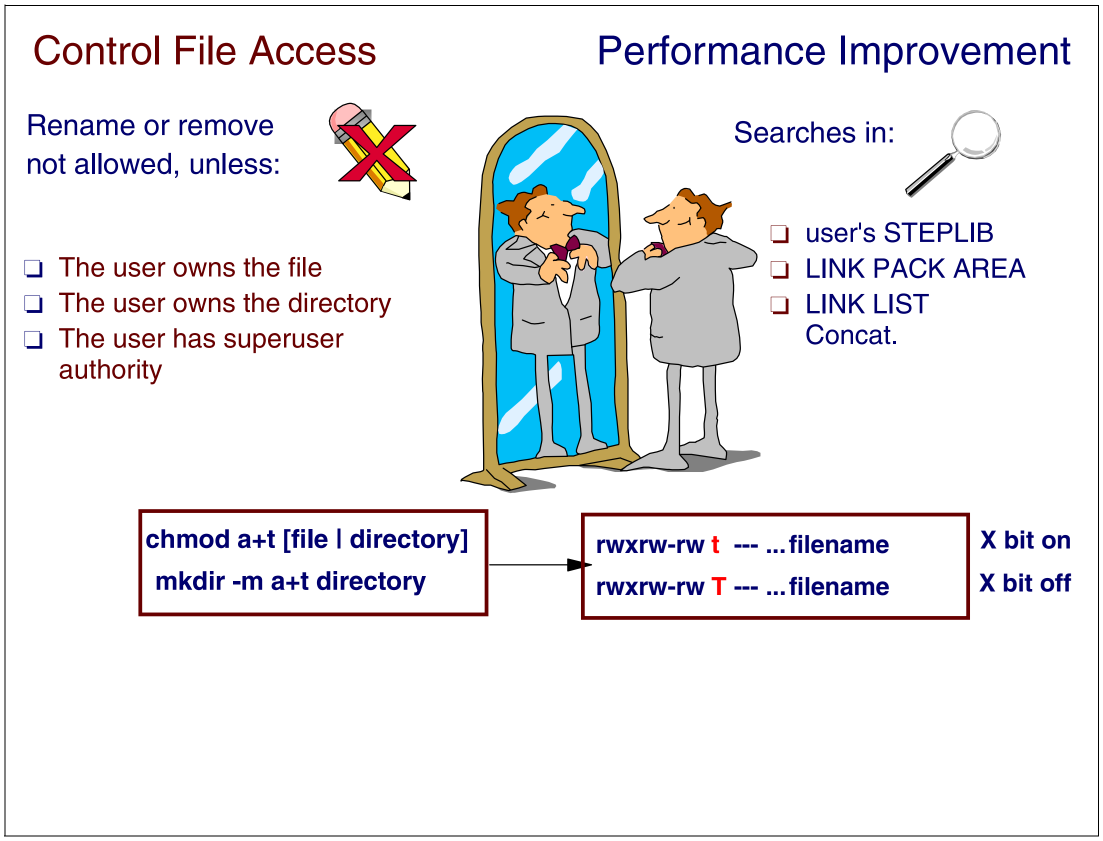

Control File Access
Performance Improvement
Rename or remove not allowed, unless:
The user owns the file
The user owns the directory
The user has superuser authority
Searches in:
user's STEPLIB
LINK PACK AREA
LINK LIST Concat.
chmod a+t [file | directory]
mkdir -m a+t directory
rwxrw-rw t --- ...filename
rwxrw-rw T --- ...filename
X bit on
X bit off

Figure 7-101 Using the sticky bit

The sticky bit is a file access permission bit that allows multiple users to share a single copy of an executable file.

Using the **mkdir**, **MKDIR**, or **chmod** command, you can set the sticky bit "on" in a directory to control permission to remove or rename files or subdirectories in the directory. When the bit is set, a user can remove or rename a file or remove a subdirectory only if one of these is true:

- The user owns the file or subdirectory.
- The user owns the directory.
- The user has superuser authority.

The sticky bit is also set for frequently used programs in the file system, to reduce I/O and improve performance. When the bit is set on, z/OS UNIX searches for the program in the user's STEPLIB, the link pack area, or the link list concatenation.

Try the list command: `ls -al`

The following two possibilities show how the sticky bit is used.

T This is the same as **t**, except that the execute bit is turned off. That means 750 has not turned on the executable bit for others, so Sticky Bit is: T

t Sticky bit is on. That means 755 has all executable bit on, so Sticky Bit is: t

The z/OS UNIX shell program (sh) is a good candidate, as follows:

- As part of the SMP/E install procedure for the shell, the shell program is placed in the root directory (/bin/sh) and the sticky bit is set.
- The shell program is also placed in SYS1.LPALIB (SH).
- During an IPL with CLPA (create LPA), the shell program will be placed in the LPA.
- Any user that invokes the shell will use the shell module in LPA and save space and I/O, since the module is not loaded into the user's address space.

If you use the **`rmdir`**, **`rename`**, **`rm`**, or **`mv`** utility to work with a file, and you receive a message that you are attempting an `operation not permitted`, check to see if the sticky bit is set for the directory the file resides in.

Note: Only someone with root authority can set the sticky bit.

7.102 Set UID/GID bit

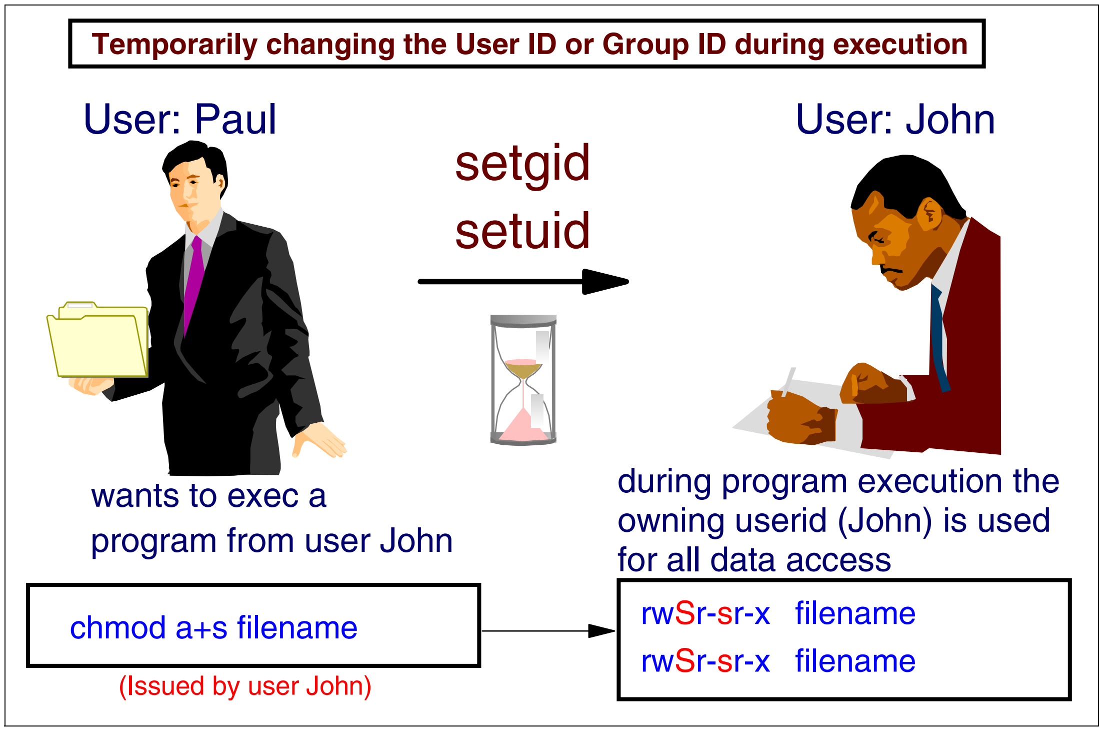

Temporarily changing the User ID or Group ID during execution
User: Paul
User: John
setgid
setuid
wants to exec a program from user John
during program execution the owning userid (John) is used for all data access
chmod a+s filename
(Issued by user John)
rwSr-sr-x filename
rwSr-sr-x filename

Figure 7-102 Using the setuid and setgid bits

If one or both of these bits are on, the effective UID and/or GID plus the saved UID and/or GID 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.

An executable file can have an additional attribute, which is displayed in the execute position (x) when you issue `ls -l`. This permission setting is used to allow a program temporary access to files that are not normally accessible to other users.

An **s** or **S** can appear in the execute permission position; this permission bit sets the effective user ID or group ID of the user process executing a program to that of the file whenever the file is run. The setuid and setgid bits are only honored for executable files.

These bits are not honored for shell script and REXX execs that reside in the file system.

s In the owner permissions section, this indicates that both the set-user-ID (S_ISUID) bit is set and the execute (search) permission is set.

In the group permissions section, this indicates that both the set-group-ID (S_ISGID) bit is set and the execute (search) permission is set.

S In the owner permissions section, this indicates the set-user-ID (S_ISUID) bit is set, but the execute (search) bit is not.

In the group permissions section, this indicates the set-group_ID (S_ISGID) bit is set, but the execute (search) bit is not.

A good example of this behavior is the `mailx` utility. A user sending something to another user on the same system is actually appending the mail to the recipient's mail file. Even though the sender does not have the appropriate permissions to do this, the mail program does.

8

Overview of TCP/IP

The *Internet*, the world's largest network, grew from fewer than 6000 participants at the end of 1986 to more than 15 million networks today. Networks have grown so quickly because they provide an important service. It is the nature of computers to generate and process information, but this information is useless unless it can be shared with the people who need it. The common thread that ties the enormous Internet together is TCP/IP network software. TCP/IP is a set of communication protocols that define how different types of computers talk to each other.

This chapter explains the basic concepts of TCP/IP, including architecture and addressing, and describes how to customize TCP/IP to use it with UNIX System Services in z/OS.

Note: Many of the figures and examples included in this chapter refer to the lab exercises for the companion class that is taught periodically. See the IBM Redbooks Web site for more information about classes offered in your geographic area.

8.1 Introduction to TCP/IP

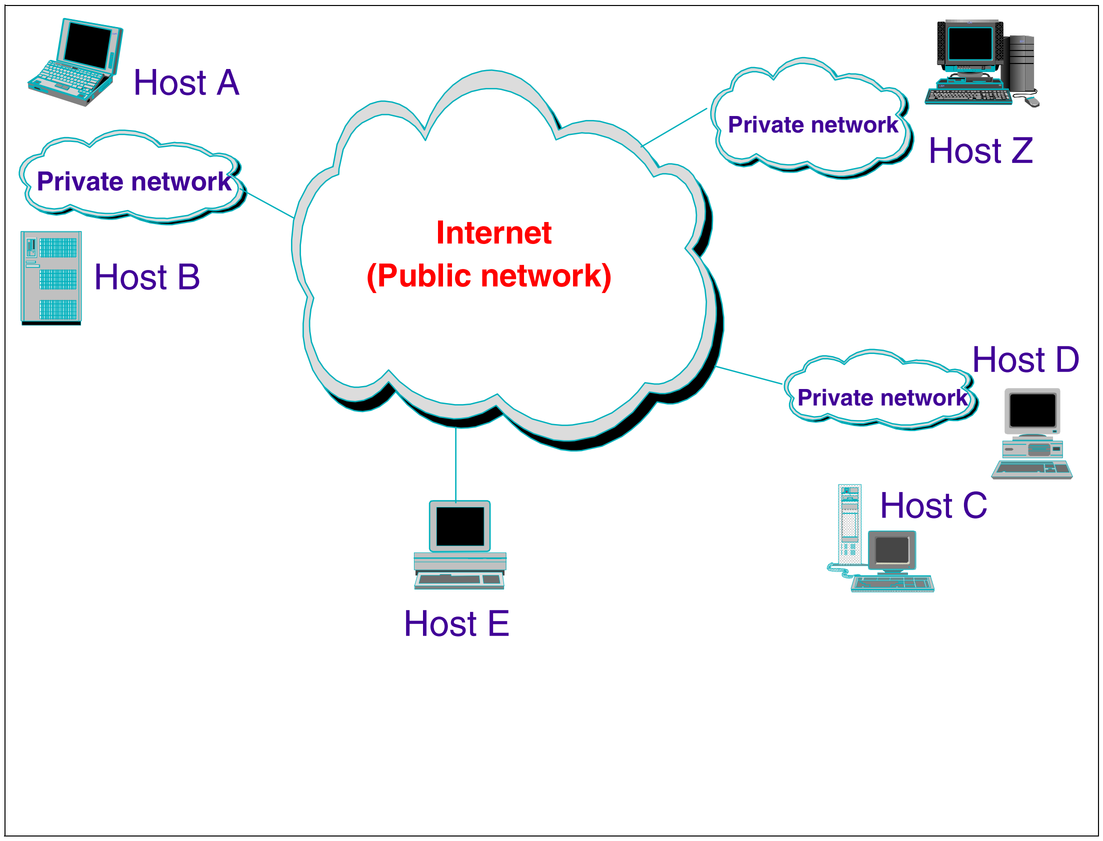


Figure 8-1 TCP/IP network

In Figure 8-1:

- Host A wants to send an e-mail to Host B.
- Host A wants to transfer data to Host D.
- Host C wants to get a Web page from Host E.
- Host Z also wants access to Host E.

These activities possible because a public network exists. Public networks are established and operated by telecommunication administrations or by Recognized Private Operating Agencies (RPOAs) for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public.

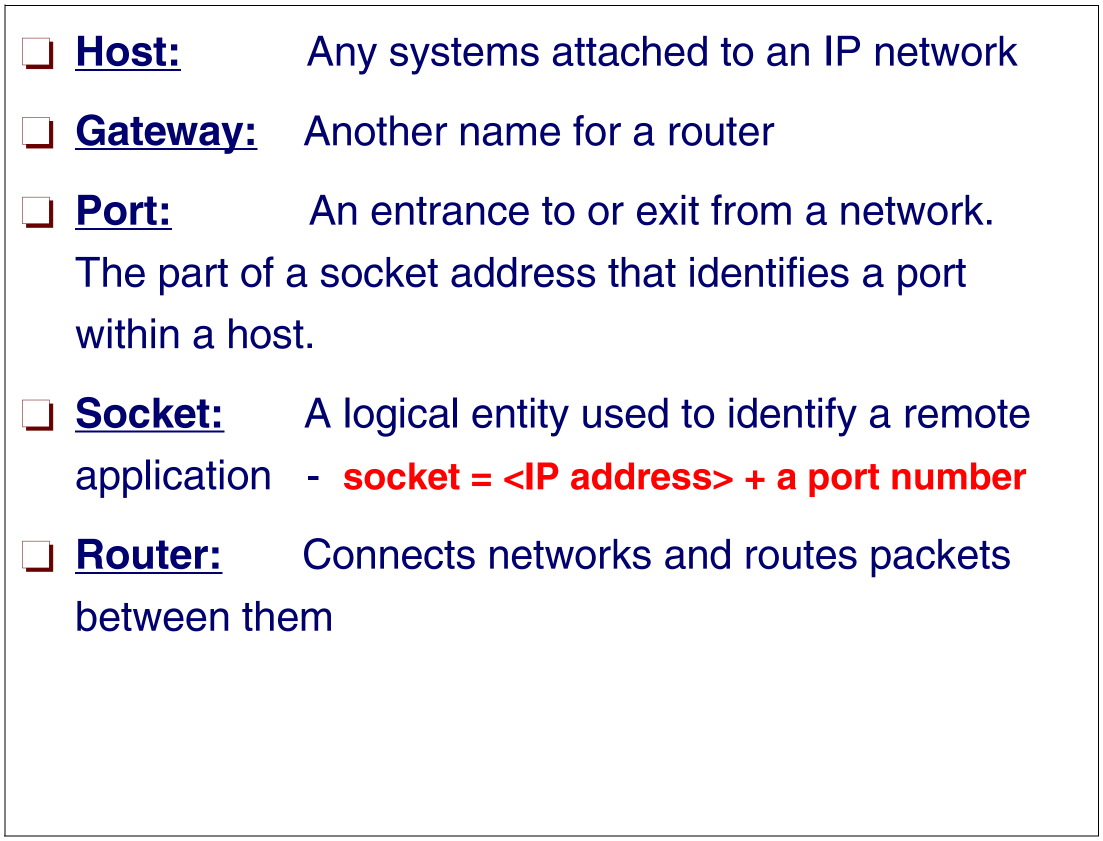
The TCP/IP protocol suite is named for two of its most important protocols: Transmission Control Protocol (TCP) and Internet Protocol (IP). Another name for it is the Internet Protocol Suite, and this is the phrase used in official Internet standards documents. The more common term TCP/IP is used to refer to the entire protocol suite.

The first design goal of TCP/IP was to build an interconnection of networks that provided universal communication services. Each physical network has its own technology-dependent communication interface, in the form of a programming interface that provides basic communication functions (*primitives*). Communication services are provided by software that runs between the physical network and the user applications and that provides a common interface for these applications, independent of the underlying physical network. The architecture of the physical networks is hidden from the user.

The second aim is to interconnect different physical networks to form what appears to the user to be one large network. Such a set of interconnected networks is called an *internetwork* or an *Internet*.

To be able to interconnect two networks, we need a computer that is attached to both networks and that can forward packets from one network to the other; such a machine is called a *router*. The term *IP router* is also used because the routing function is part of the IP layer of the TCP/IP protocol suite.

8.2 TCP/IP terminology



- **Host:** Any systems attached to an IP network
- **Gateway:** Another name for a router
- **Port:** An entrance to or exit from a network. The part of a socket address that identifies a port within a host.
- **Socket:** A logical entity used to identify a remote application - **socket = <IP address> + a port number**
- **Router:** Connects networks and routes packets between them

Figure 8-2 TCP/IP terminology

The following terminology is commonly used to describe a TCP/IP environment:

Host In the Internet suite of protocols, this is an end system. The end system can be any workstation; it does not have to be a mainframe.

Gateway A functional unit that interconnects two computer networks with different network architectures. A *gateway* connects networks or systems of different architectures. A *bridge* interconnects networks or systems with the same or similar architectures. In TCP/IP, this is a synonym for router.

Port Each process that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports. A port is a 16-bit number, used by the host-to-host protocol to identify to which higher level protocol or application program (process) it must deliver incoming messages. There are two types of ports:

Well-known Well-known ports belong to standard servers, for example Telnet uses port 23. Well-known port numbers range between 1 and 1023 (prior to 1992, the range between 256 and 1023 was used for UNIX-specific servers). Well-known port numbers are typically odd, because early systems using the port concept required an odd/even pair of ports for duplex operations. Most servers require only a single port. The well-known ports are controlled and assigned by the Internet central authority (IANA) and on most systems can only be used by system processes or by programs executed by privileged users. The reason for well-known ports is to allow clients to be able to find servers without configuration information.

Ephemeral Clients do not need well-known port numbers because they initiate communication with servers and the port number they are using is contained in the UDP datagrams sent to the server. Each client process is allocated a port number as long as it needs it by the host it is running on. Ephemeral port numbers have values greater than 1023, normally in the range 1024 to 65535.

Socket An endpoint for communication between processes or application programs. A synonym for port.

Socket address The address of an application program that uses the socket interface on the network. In Internet format, it consists of the IP address of the socket's host and the port number of the socket. The application program is usually not aware of the structure of the address.

Socket interface A Berkeley Software Distribution (BSD) application programming interface (API) that allows users to easily write their own programs.

Router A router interconnects networks at the internetwork layer level and routes packets between them. The router must understand the addressing structure associated with the networking protocols it supports and make decisions on whether, or how, to forward packets. Routers are able to select the best transmission paths and optimal packet sizes. The basic routing function is implemented in the IP layer of the TCP/IP protocol stack, so any host or workstation running TCP/IP over more than one interface could, in theory and also with most of today's TCP/IP implementations, forward IP datagrams. However, dedicated routers provide much more sophisticated routing than the minimum functions implemented by IP.

8.3 IP addressing

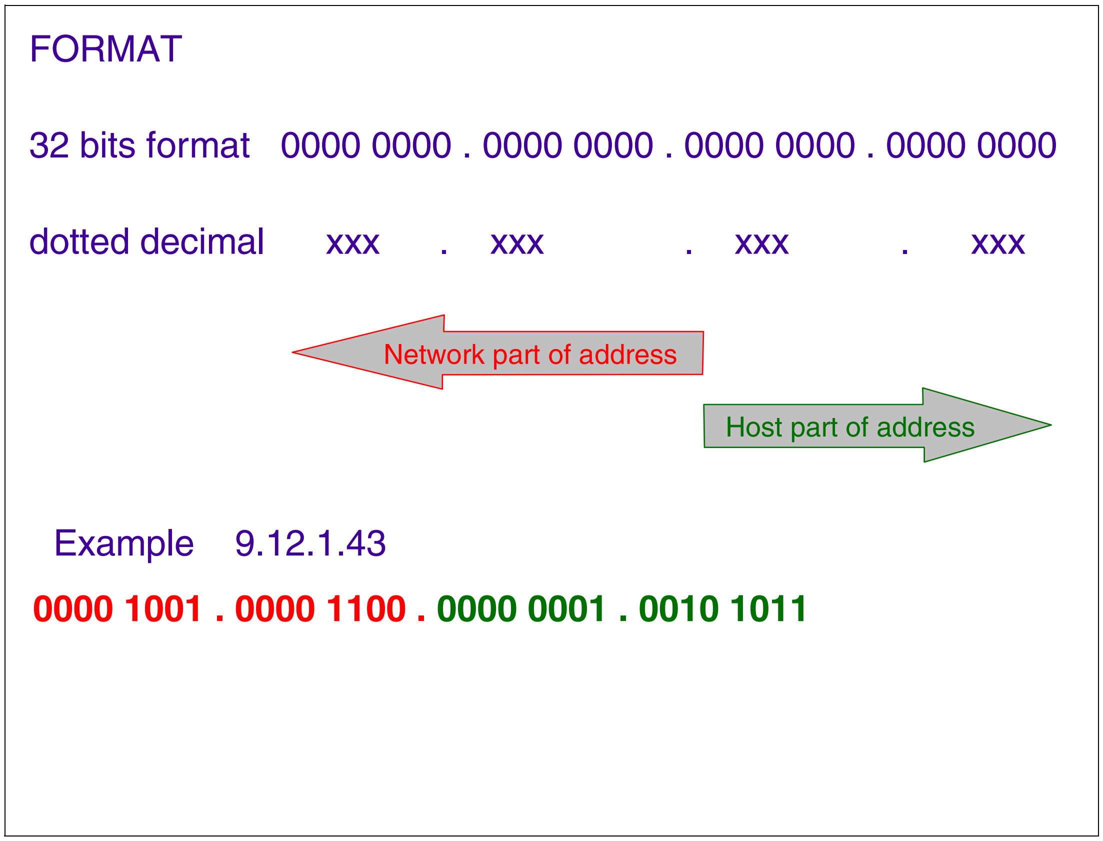


Figure 8-3 IP addressing examples

To be able to identify a host on the Internet, each host is assigned an address, called the IP address, or Internet address. When the host is attached to more than one network, it is called multi-homed and it has one IP address for each network interface.

IP addressing

An IP address is represented by a 32-bit unsigned binary value which is usually expressed in a dotted decimal format. For example, 9.12.1.43 is a valid Internet address. The numeric form is used by the IP software. The mapping between the IP address and an easier-to-read symbolic name, for example myhost.ibm.com®, is done by a Domain Name System. We first look at the numeric form, which is called the IP address.

The Internet Protocol uses IP addresses to specify source and target hosts on the Internet. Each byte is represented by its decimal form:

```
9.12.1.43 = 0000 1001 . 0000 1100 .  0000 0001 . 0010 1011
```

Each host must have a unique Internet address to communicate with other hosts on the Internet. The network address part of the IP address is centrally administered by the Internet Network Information Center (the InterNIC) and is unique throughout the Internet. Each IP address is made up of two logical addresses:

```
IP address = <network address> <host address>
```

where:

network address Represents a specific physical network within the Internet.

host address Specifies an individual host within the physical network identified by the network address.

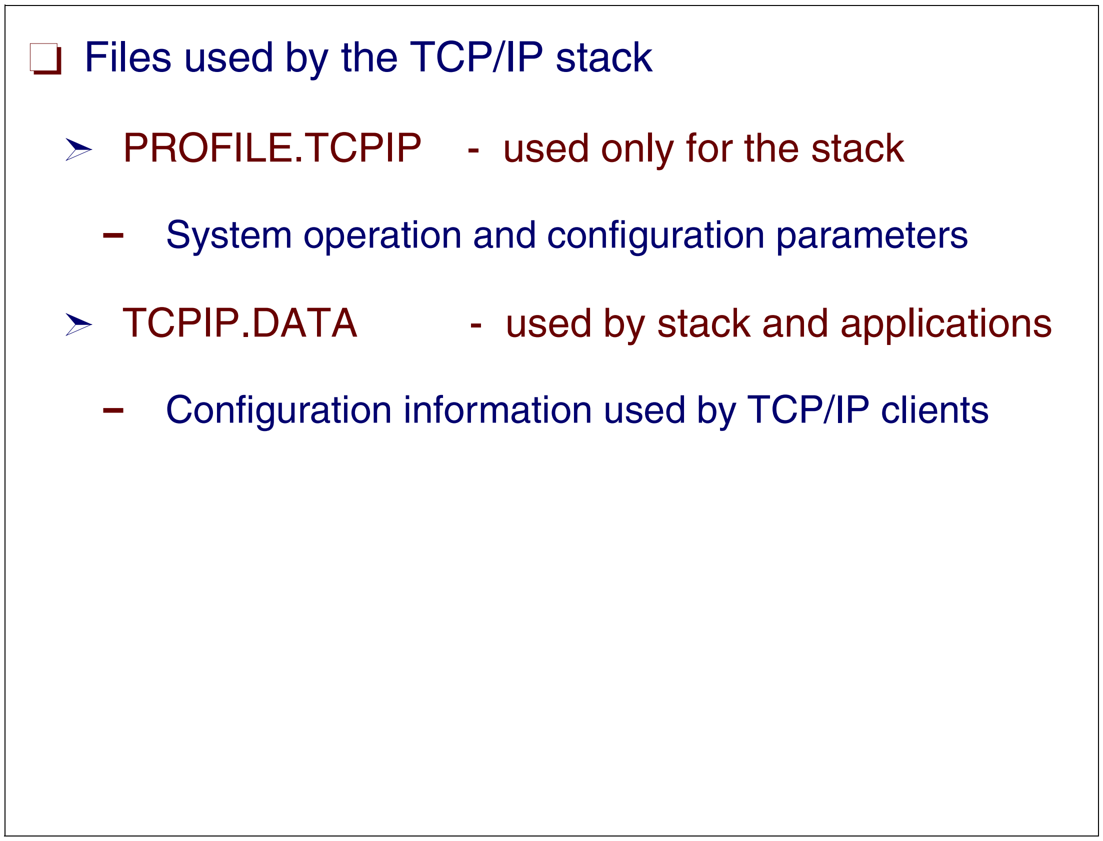
For example, 9.12.1.43 is an IP address with 9.12 being the network address and 1.43 being the host address.

A *subnet mask* is used to differentiate the network address and host address.

The first bits of the IP address specify how the rest of the address should be separated into its network and host part.

The Internet Protocol moves data between hosts in the form of *datagrams*. Each datagram is delivered to the address contained in the Destination Address of the datagram's header.

8.4 Configuration files used by TCP/IP



- Files used by the TCP/IP stack
  - PROFILE.TCPIP - used only for the stack
    - System operation and configuration parameters
  - TCPIP.DATA - used by stack and applications
    - Configuration information used by TCP/IP clients

Figure 8-4 Configuration files used by TCP/IP initialization

Two configuration files are used by the TCPIP stack, PROFILE.TCPIP and TCPIP.DATA.

PROFILE.TCPIP is used only for the configuration of the TCPIP stack. During initialization of the TCPIP stack, also referred to as the TCPIP address space, system operation and configuration parameters for the TCPIP stack are read from the configuration file PROFILE.TCPIP.

TCPIP.DATA is used during configuration of both the TCPIP stack and applications. This data set, TCPIP.DATA, is used to specify configuration information required by TCP/IP client programs.

8.5 Customize TCP/IP profile data set

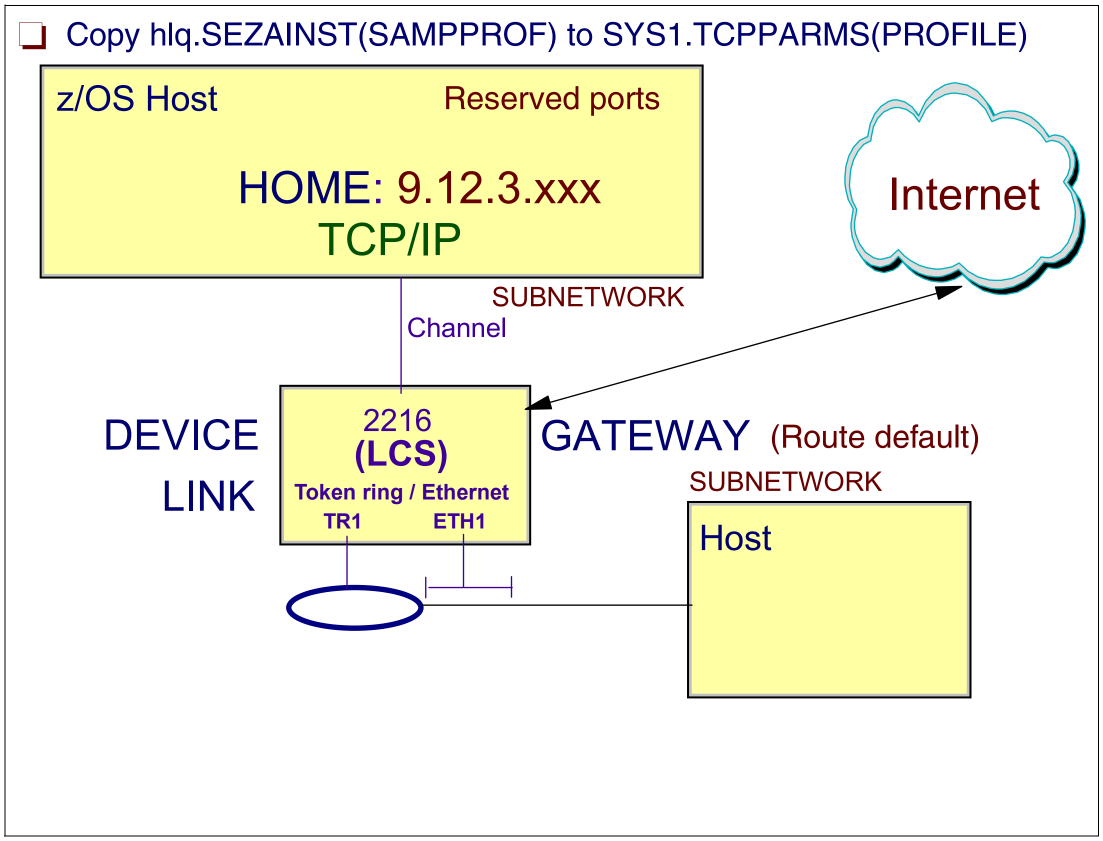


Figure 8-5 Customizing the TCP/IP profile data set

This is a list of PROFILE statements that need to be customized to define your environment. The IP address of the HOME statement for this host, as well as the GATEWAY value such as subnet mask, subnet and DEFAULTNET (or default gateway), can be obtained from your network administrator.

A sample of the PROFILE data sets is provided in hlq.SEZAINST(SAMPPROF), which you can copy to SYS1.TCPPARMS(PROFILE).

TCP/IP reads the parameters from the TCP/IP profile data set.

The parameters that need to be changed are:

AUTOLOG	Uncomment FTPD or any other daemons that need to be activated.
DEVICE	Provide the z/OS address network interface. It could be the OSA address, CTC, IBM 2216 router or any other supported network device.
LINK	Provide the description of the network interface.
HOME	Specify the IP address of the z/OS system.
Begin route	Specify the IP address of the net and subnet to which this host belongs.
Route default	Specify the default gateway IP address. Usually, this is the IP address of the network router to which this host is attached.

8.6 Customize TCPDATA

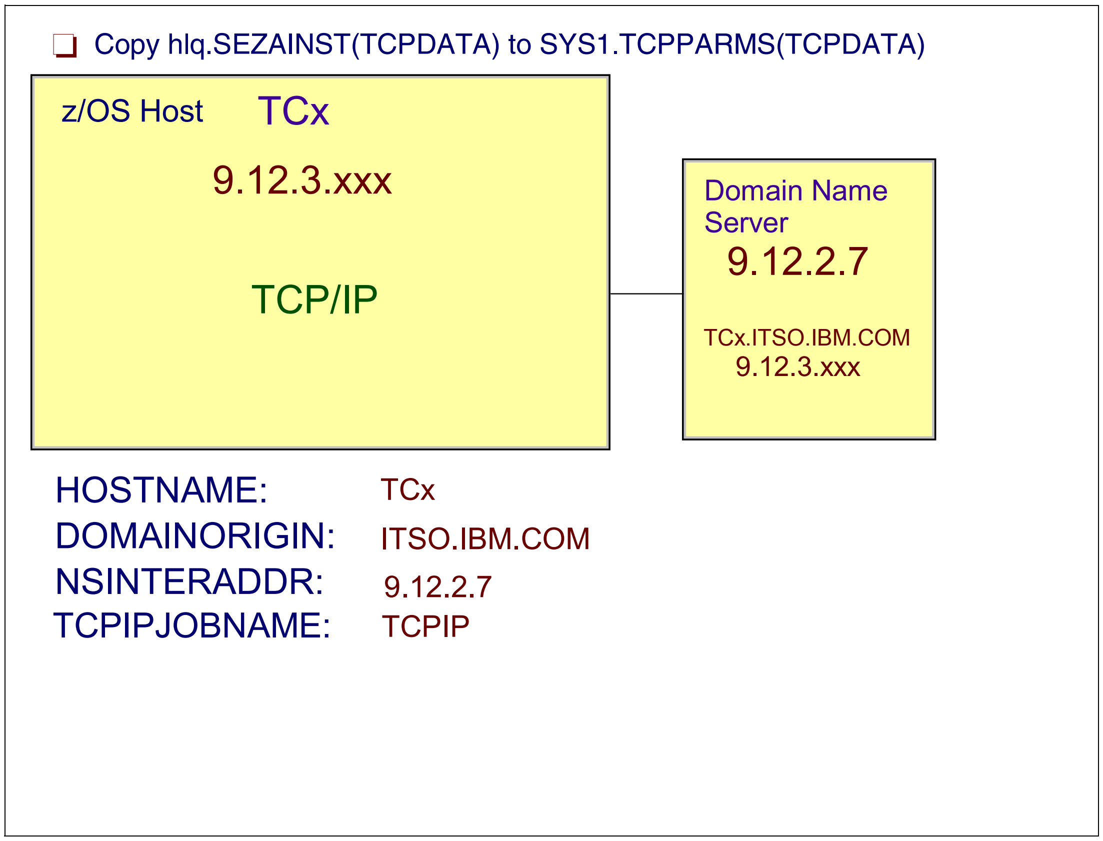


Figure 8-6 Customizing the TCPDATA data set

Parameters that need to be customized for TCPDATA are indicated in Figure 8-6.

For the DOMAINORIGIN and NSINTERADDR statements, the values can be obtained from your network administrator.

The SYSTCPD DD explicitly identifies which data set is to be used to obtain the parameters defined by TCPIP.DATA. The SYSTCPD DD statement should be placed in the TSO/E logon procedure or in the JCL of any client or server executed as a background task. The data set can be any sequential data set or a member of a partitioned data set (PDS).

```
//SYSTCPD  DD DSN=SYS1.TCPPARMS(TCPDATA),DISP=SHR
```

Parameters that need to be changed for the TCPDATA file are:

- TCPIPJOBNAME specifies the TCPIP started task jobname.
- HOSTNAME specifies the hostname (SYSNAME on IEASYSxx) or IEASYMxx.
- DATASETPREFIX specifies the hlq you have selected before.

Other optional parameters are:

- DOMAINORIGIN specifies your domain (ITSO.IBM.COM).
- NSINTERADDR specifies your name server IP address.

8.7 z/OS IP search order

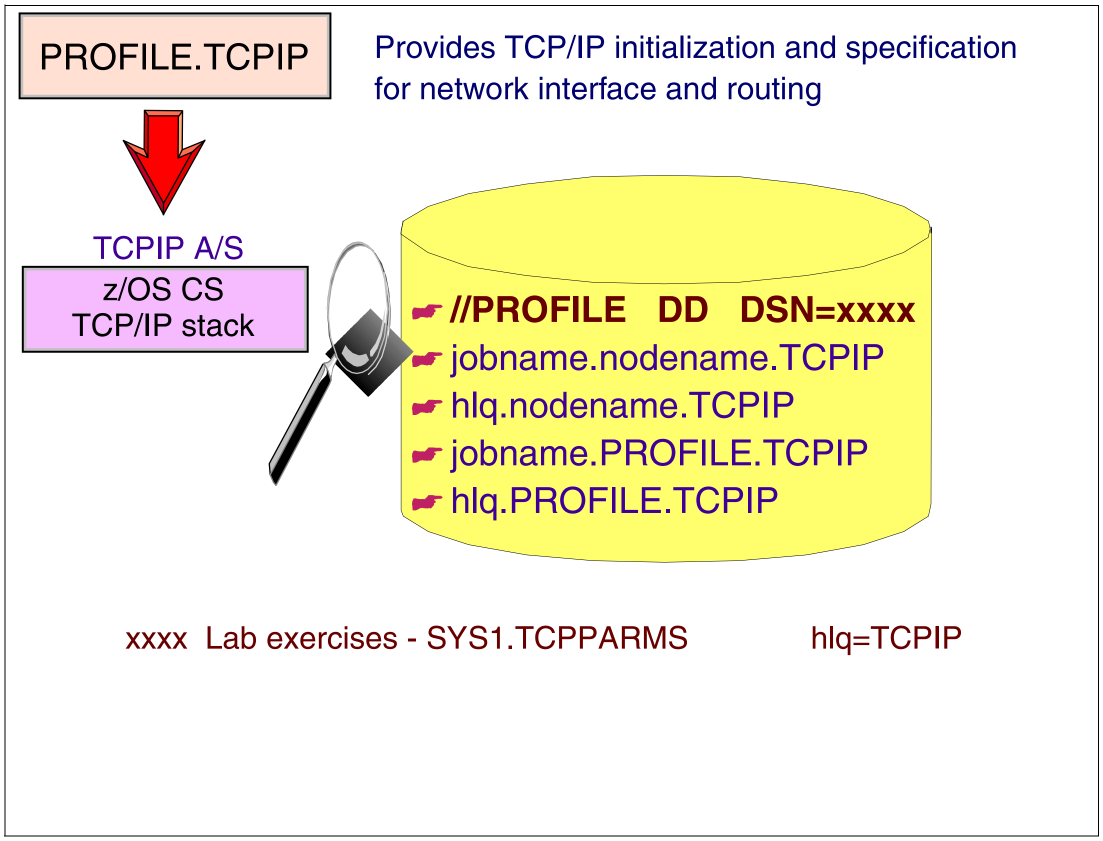


Figure 8-7 Search order used during TCP/IP initialization

During the initialization of the TCP/IP stack, system configuration and configuration parameters for the TCP/IP stack are read from the configuration profile PROFILE.TCPIP.

The search order used by TCP/IP to find PROFILE.TCPIP data sets involves both explicit and dynamic data set allocation.

When TCP/IP starts, it looks for the PROFILE data set in the following order:

- //PROFILE DD explicitly specified in the PROFILE DD statement of the TCP/IP started task procedure.
- jobname.nodename.TCPIP data set
- hlq.nodename.TCPIP data set
- jobname.PROFILE.TCPIP data set
- hlq.PROFILE.TCPIP data set

The search stops if one of these data sets is found.

8.8 z/OS IP search order (2)

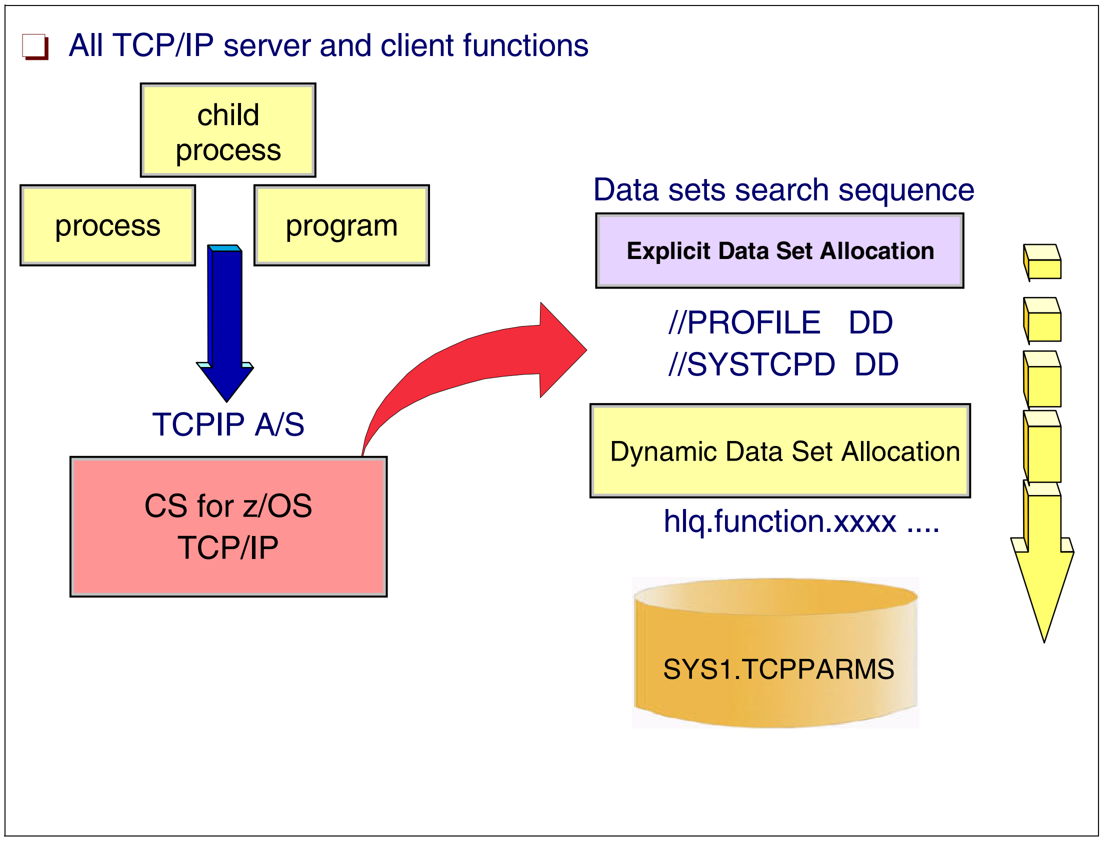

All TCP/IP server and client functions
child process
process
program
TCPIP A/S
CS for z/OS TCP/IP
Data sets search sequence
Explicit Data Set Allocation
//PROFILE DD
//SYSTCPD DD
Dynamic Data Set Allocation
hlq.function.xxxx ....
SYS1.TCPPARMS

Figure 8-8 Search order for z/OS UNIX applications

The CS for z/OS environment consists of the CS for z/OS stack, CS for z/OS applications (z/OS UNIX System Services applications such as Telnetd, FTPD...) and the z/OS TCP/IP native MVS applications.

The TCP/IP stack and set of applications have some common configuration files, but they also use configuration files that are different.

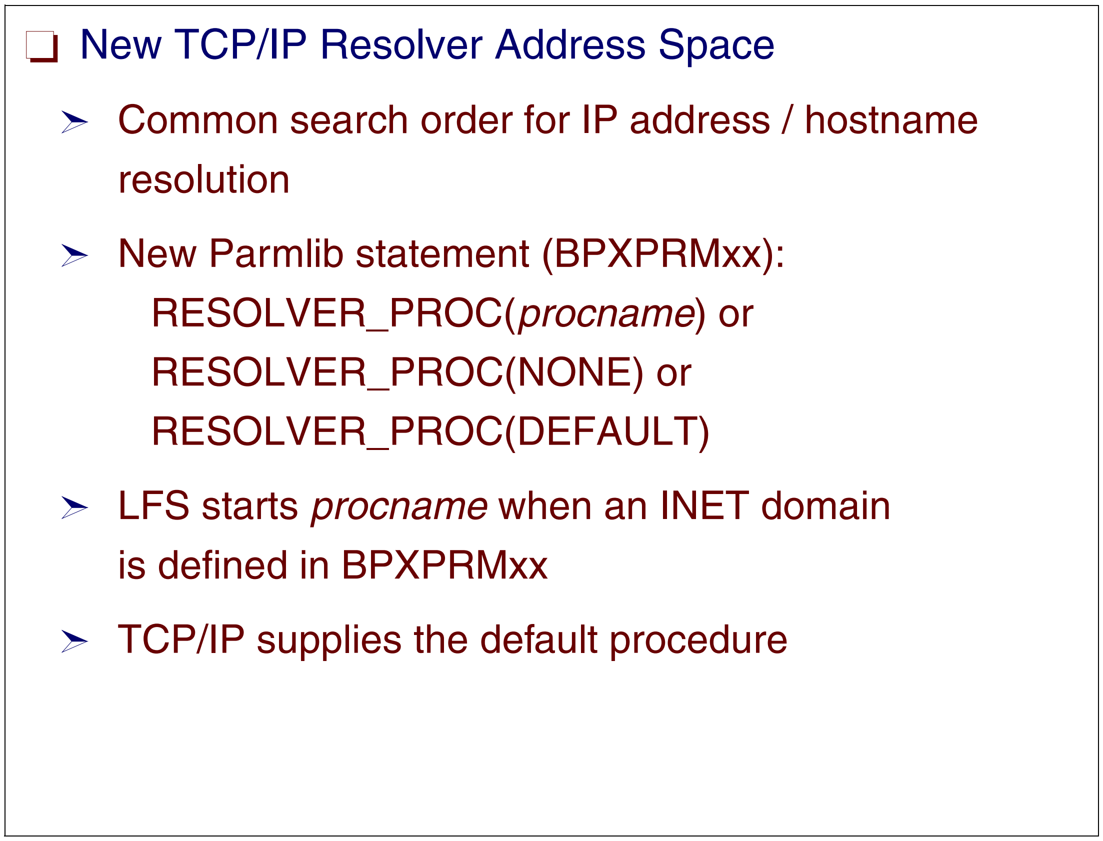
Different configuration files may be used for a TCP/IP stack where there is a need to understand the search order for each z/OS UNIX application.

The search order is applied to any configuration file, and the search ends with the first file found, as follows:

- **Explicit Data Set Allocation** consists of those data sets that you specify through the use of DD statements in JCL procedures.
- **Dynamic Data Set Allocation** consists of multiple versions of a data set, each having a different high-level qualifier or middle-level qualifier, and some data sets that can only be dynamically allocated by TCP/IP (they cannot be allocated using DD statements in JCL).

There is a naming convention for dynamically allocated data sets.

8.9 TCPDATA search order



❑ New TCP/IP Resolver Address Space

➢ Common search order for IP address / hostname resolution

➢ New Parmlib statement (BPXPRMxx):
RESOLVER_PROC(*procname*) or
RESOLVER_PROC(NONE) or
RESOLVER_PROC(DEFAULT)

➢ LFS starts *procname* when an INET domain is defined in BPXPRMxx

➢ TCP/IP supplies the default procedure

Figure 8-9 Search order for TCPDATA using the resolver address space

TCPIP.DATA is used during configuration of both the TCP/IP stack and applications; the search order to find the TCPIP.DATA data set is the same for both the TCP/IP stack and applications. The search order used by TCP/IP and applications is as follows:

- MVS data set or HFS file specified by the environment variable RESOLVER_CONFIG
- /etc/resolv.conf
- //SYSTCPD DD
- jobname.TCPIP.DATA data set
- SYS1.TCPPARMS(TCPDATA)
- hlq.TCPIP.DATA data set

The search stops if one of these data sets is found.

A name resolver converts a TCP/IP hostname to an IP address, or vice versa. There are several name resolvers in TCP/IP, one in Language Environment, and one in CICS, and multiple resolver libraries exist. There are MVS native and Language Environment resolver APIs. The search order for TCPIP.DATA varies across resolver libraries. This can lead to an inconsistent name resolution process. It makes it difficult to provide resolver enhancements in a consistent and timely manner.

A new TCP/IP resolver address space supports the consolidation of the many ways to resolve host names or IP addresses. It provides for both global and local user settings to be configured.

LFS is responsible for starting the new TCP/IP resolver address space. The new address space is started during IPL. In order for LFS to start this new resolver address space, the system must be configured with an AF_INET socket file system domain name in BPXPRMxx.

There are new GetHostByName and GetHostByAddr functions in the LFS. These functions are assembler macro callable services BPX1GHN and BPX1GHA. These new callable services will now be used by TCP/IP and Language Environment. They will actually call TCP/IP code if the resolver address space is started.

BPXPRMxx parmlib member

We recommend that you update the BPXPRMxx parmlib member and use this to start the resolver address space. Using the BPXPRMxx member and OMVS initialization to start the resolver helps ensure that the resolver API is available before any /etc/rc or COMMNDxx parmlib members are executed.

The RESOLVER_PROC statement in parmlib member BPXPRMxx is used to start the resolver during z/OS UNIX System Services initialization.

The statement specifies how the resolver address space is processed during UNIX System Services initialization. The resolver address space is used by TCP/IP applications for name-to-address or address-to-name resolution.

```
RESOLVER_PROC(procname|DEFAULT|NONE)
```

The `procname` is the name of the address space for the resolver and the procedure member name in SYS1.PROCLIB.

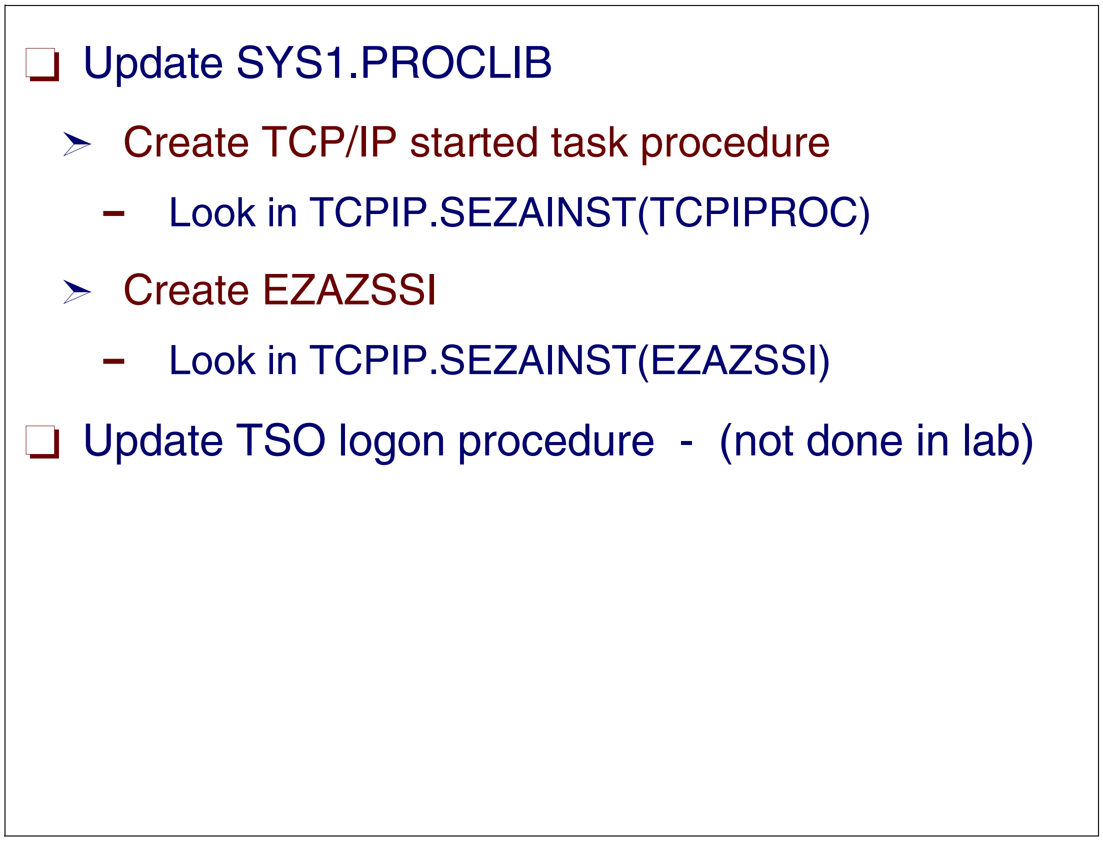
DEFAULT causes an address space named RESOLVER to start. This also happens if the RESOLVER_PROC statement is not specified in the BPXPRMxx.

NONE specifies that no address space is to be started.

The MVS operator console command is:

```
D OMVS,OPTIONS
```

8.10 Customize TCP/IP procedure



- ❑ Update SYS1.PROCLIB
  - ➢ Create TCP/IP started task procedure
    - – Look in TCPIP.SEZAINST(TCPIPROC)
  - ➢ Create EZAZSSI
    - – Look in TCPIP.SEZAINST(EZAZSSI)
- ❑ Update TSO logon procedure - (not done in lab)

Figure 8-10 Customizing the TCP/IP procedure

Create the PROCLIB member, as follows:

- TCPIP started task procedure - a sample is provided in hlq.SEZAINST(TCPIPROC)
- EZAZSSI procedure to start TCP Subsystem Interface

Add procedure EZAZSSI to your system PROCLIB. A sample of this procedure is located in the data set hlq.SEZAINST (where hlq is the high-level qualifier for the TCP/IP product data sets in your installation).

```
//EZAZSSI    PROC   P=''
//STARTVT    EXEC   PGM=EZAZSSI,PARM=&P
//STEPLIB    DD     DSN=hlq.SEZALINK,DISP=SHR
//           DD     DSN=hlq.SEZATCP,DISP=SHR
```

You can remove the STEPLIB DD if these data sets are defined on LNKLSTxx, as follows:

- Modify your TSO/E logon procedure:
  - To include hlq.SEZAHELP in //SYSHELP DD
  - To include hlq.SEZAMENU in //ISPMLIB DD
  - To include hlq.SEZAPENU in //ISPTLIB DD and //ISPPLIB DD

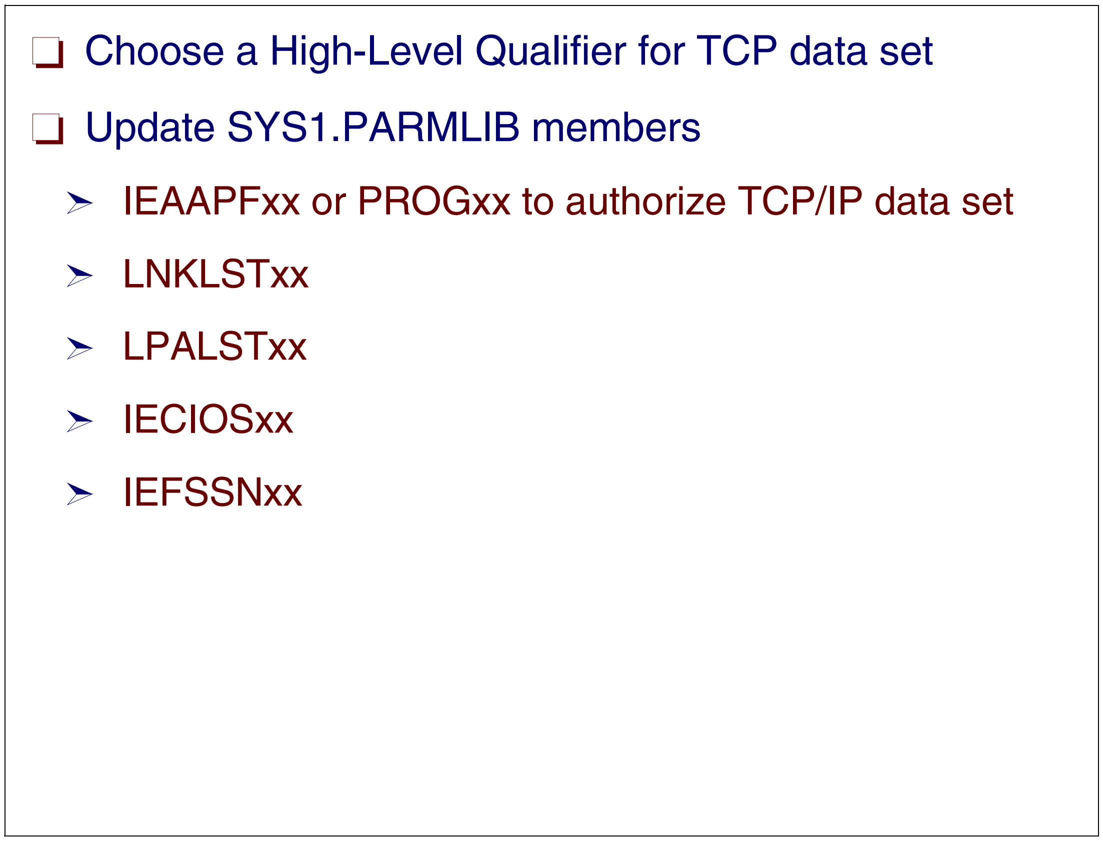
  - Optionally, add a //SYSTCPD DD to point to the TCPDATA data set in order to use TCP/IP client functions and some administrative functions such as OBEYFILE under TSO/E.

    SYSTCPD explicitly identifies the data set used to obtain parameters defined by TCPIP.DATA.

- The SYSTCPD statement should be placed in the TSO/E logon procedure or in the JCL of any client or server executed as a background task. The data set can be any sequential data set or a member of a partitioned data set (PDS). TSO client functions can be directed against any of a number of TCP/IP stacks. Obviously, the client function must be able to find the TCPIP.DATA appropriate to the stack of interest at any one time. Two methods are available for finding the relevant TCPIP.DATA:
  - Add a SYSTCPD DD statement to your TSO logon procedure. The issue with this approach is that a separate TSO logon procedure per stack is required, and users have to log off TSO and log on again using another TSO logon procedure in order to switch from one stack to another.
  - Use one common TSO logon procedure without a SYSTCPD DD statement. Before a TSO user starts any TCP/IP client programs, the user has to issue a TSO ALLOC command wherein the user allocates a TCPIP.DATA data set to DDname SYSTCPD. To switch from one stack to another, the user simply has to de-allocate the current SYSTCPD allocation and allocate another TCPIP.DATA data set.

  Combine the first and second methods. Use one logon procedure to specify a SYSTCPD DD for a default stack. To switch stacks, issue TSO ALLOC to allocate a new SYSTCPD. To switch back, issue TSO ALLOC again with the name that was on the SYSTCPD DD in the logon procedure. The disadvantage to this approach is that the name that was on the SYSTCPD DD is “hidden” in the logon procedure and needs to be retrieved or remembered.

8.11 Customizing parmlib members for TCP/IP



- Choose a High-Level Qualifier for TCP data set
- Update SYS1.PARMLIB members
  - IEAAPFxx or PROGxx to authorize TCP/IP data set
  - LNKLSTxx
  - LPALSTxx
  - IECIOSxx
  - IEFSSNxx

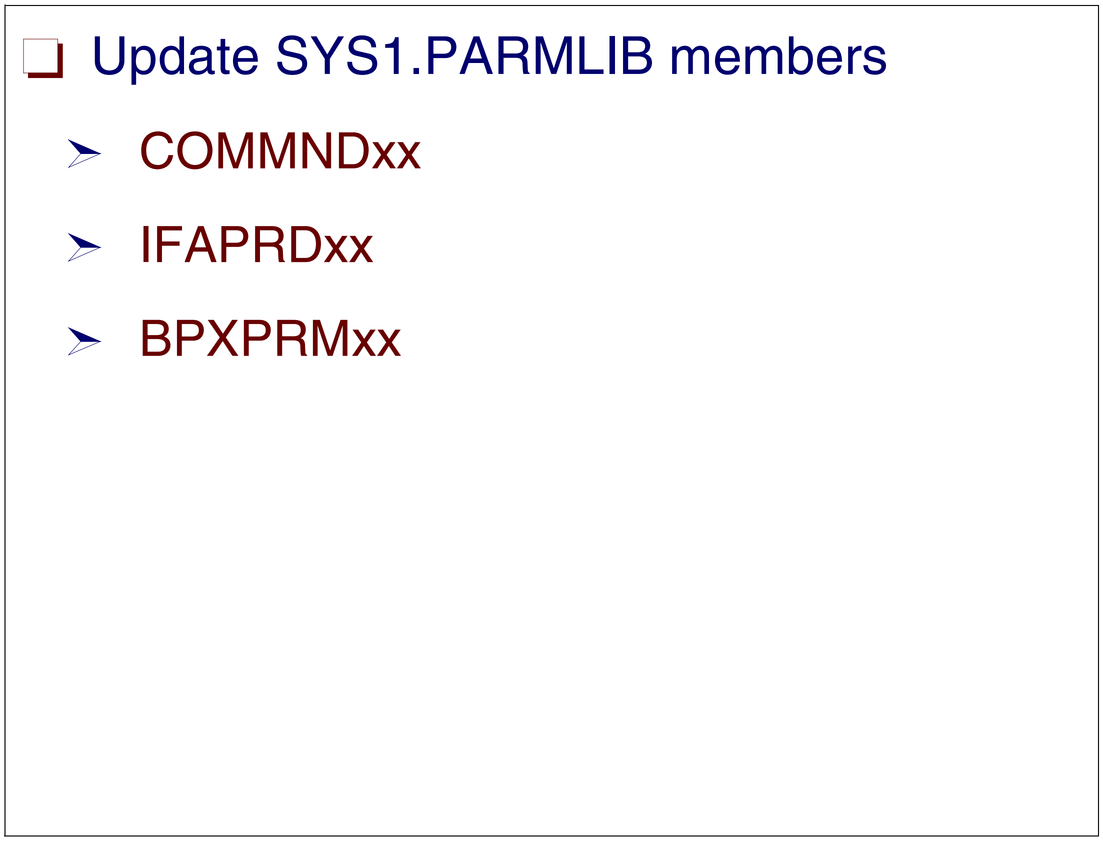
Figure 8-11 Parmlib members to customize for TCP/IP

Steps for customizing TCP/IP are as follows:

- Choose a High-Level Qualifier (hlq).
- TCP/IP uses dynamic allocation with this hlq to get parameters from several TCP/IP data sets. The DATASETPREFIX statement in TCPIP.DATA can be used to override the default hlq. However, it is used as the last step in the search order for most configuration files.
- Update the following PARMLIB members:

IEAAPFxx/PROGxx	Authorizes the following libraries:
	`hlq.SEZATCP`
	`hlq.SEZADSIL`
	`hlq.SEZALOAD`
	`hlq.SEZALNK2`
	`hlq.SEZALPA`
	`hlq.SEZAMIG`
LNKLSTxx	hlq.SEZALOAD
LPALSTxx	hlq.SEZALPA and hlq.SEZATCP
IECIOSxx	Use to disable the MIH processing for communication device used by TCP/IP
	`Set MIH TIME=00:00,DEV=(cuu-cuu)`
IEFSSNxx	Define subsystems to use restartable VMCF and TNF
	`SUBSYS SUBNAME(TNF)`
	`SUBSYS SUBNAME(VMCF)`

8.12 Parmlib members to customize for TCP/IP


❑ Update SYS1.PARMLIB members

- ➢ COMMNDxx
- ➢ IFAPRDxx
- ➢ BPXPRMxx

Figure 8-12 Parmlib members to customize for TCP/IP

Update the following PARMLIB members:

COMMNDxx Add the following command to start the SSI

```
COM='S EZAZSSI,P=sys_name'
```

Where sys_name is the SYSNAME in IEASYSxx or specified in IEASYMxx using the SYSDEF statement.

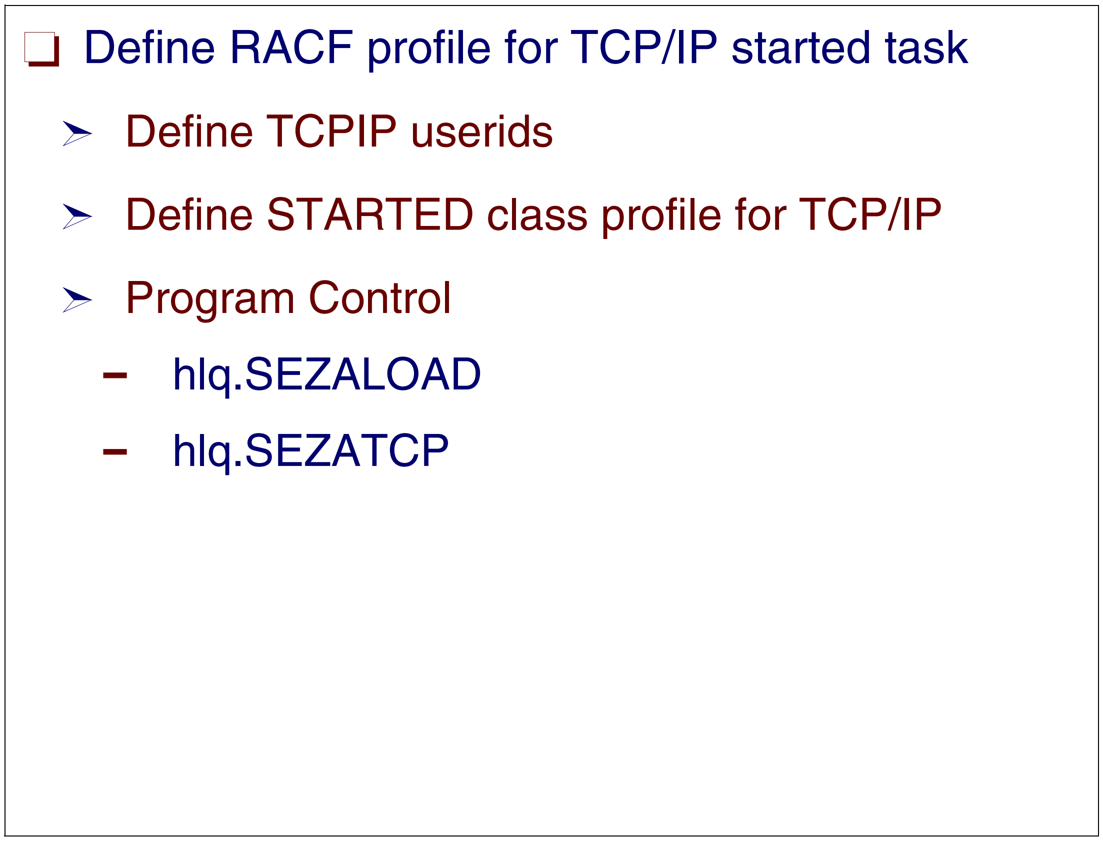
IFAPRDxx Enable TCP/IP base by adding this:

```
NAME(z/OS) ID(5647-A01)
```

BPXPRMxx Activate TCP/IP support for transport provider as follows:

```
FILESYTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
    NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETSR(10000)
        TYPE(INET)
```

8.13 RACF customization for TCP/IP



- Define RACF profile for TCP/IP started task
  - Define TCPIP userids
  - Define STARTED class profile for TCP/IP
  - Program Control
    - hlq.SEZALOAD
    - hlq.SEZATCP

Figure 8-13 Customizing TCP/IP with RACF profiles

You need to define a RACF user ID with an OMVS segment for TCPIP, PORTMAP, NFS, and FTPD as shown:

PROCNAME	RACF user ID	UID	RACF Group	GID	Trusted
TCPIP	TCPIP	0	OMVSGRP	1	No
PORTMAP	TCPIP	0	OMVSGRP	1	No
NFS	TCPIP	0	OMVSGRP	1	No
FTPD	TCPIP	0	OMVSGRP	1	No
INETD	OMVSKERN	0	OMVSGRP	1	No

You also need to define the following data sets to program control:

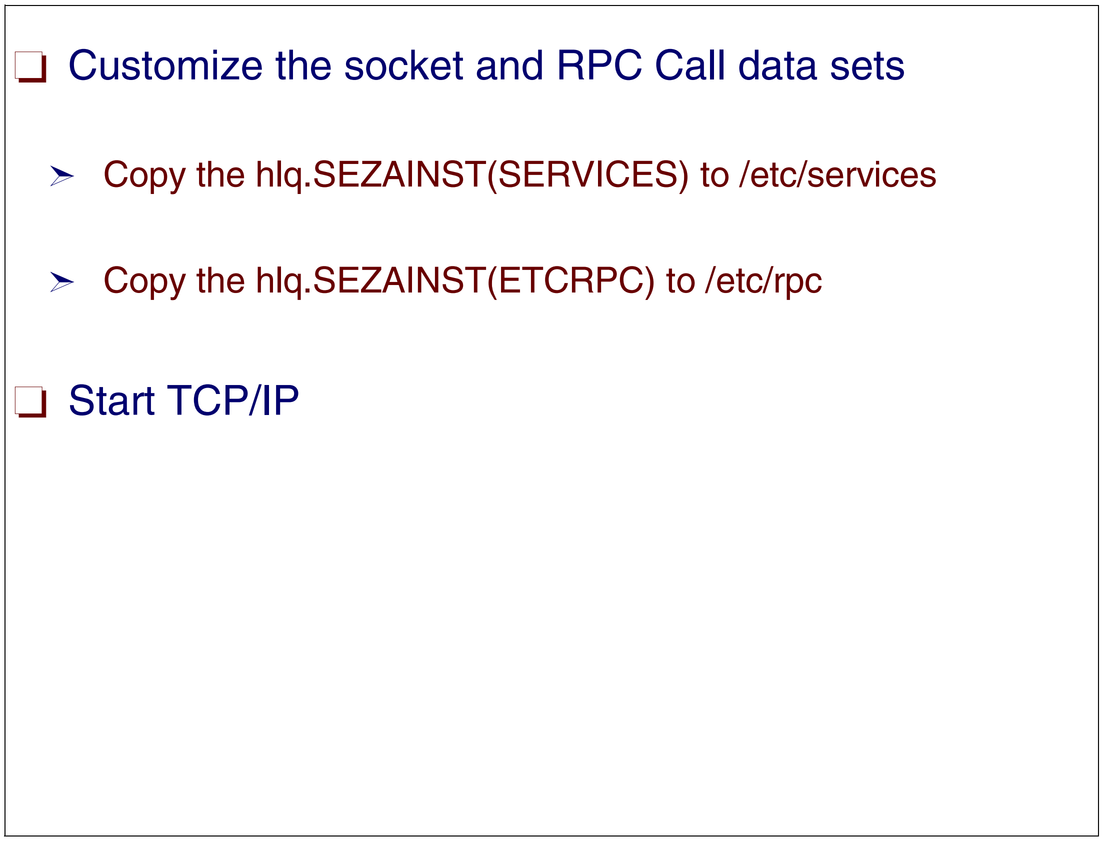
- SYS1.LINKIB
- hlq.SEZALOAD
- hlq.SEZATCP
- CEE.SCEERUN (or Language Environment Run-time Modules)

In a z/OS UNIX environment, there are additional security concerns related to the HFS and the loading of programs that are considered trusted. Program control facilities in RACF and z/OS provide a mechanism for ensuring that the z/OS program loading process has the same security features that APF authorization provides in the native MVS environment.

It is recommended that you enable program control in your installation. If you define the BPX.DAEMON Facility Class, then you must enable program control for certain CS for z/OS load libraries. Review the section on Program Control in the z/OS UNIX MVS Planning publication to decide whether program control is appropriate for your installation.

When you use program control, make sure that all load modules that are loaded into an address space come from controlled libraries. If the MVS contents supervisor loads a module from a non-controlled library, the address space becomes dirty and loses its authorization. To prevent this from happening, define all the libraries from which load modules can be loaded as program controlled. At a minimum, this should include the C runtime library, the TCP/IP for MVS SEZALOAD and SEZATCP libraries, and SYS1.LINKLIB.

8.14 HFS data set to customize for TCP/IP



❑ Customize the socket and RPC Call data sets

➢ Copy the hlq.SEZAINST(SERVICES) to /etc/services

➢ Copy the hlq.SEZAINST(ETCRPC) to /etc/rpc

❑ Start TCP/IP

Figure 8-14 HFS data sets t customize for TCP/IP

SERVICES and RPC data set are as follows:

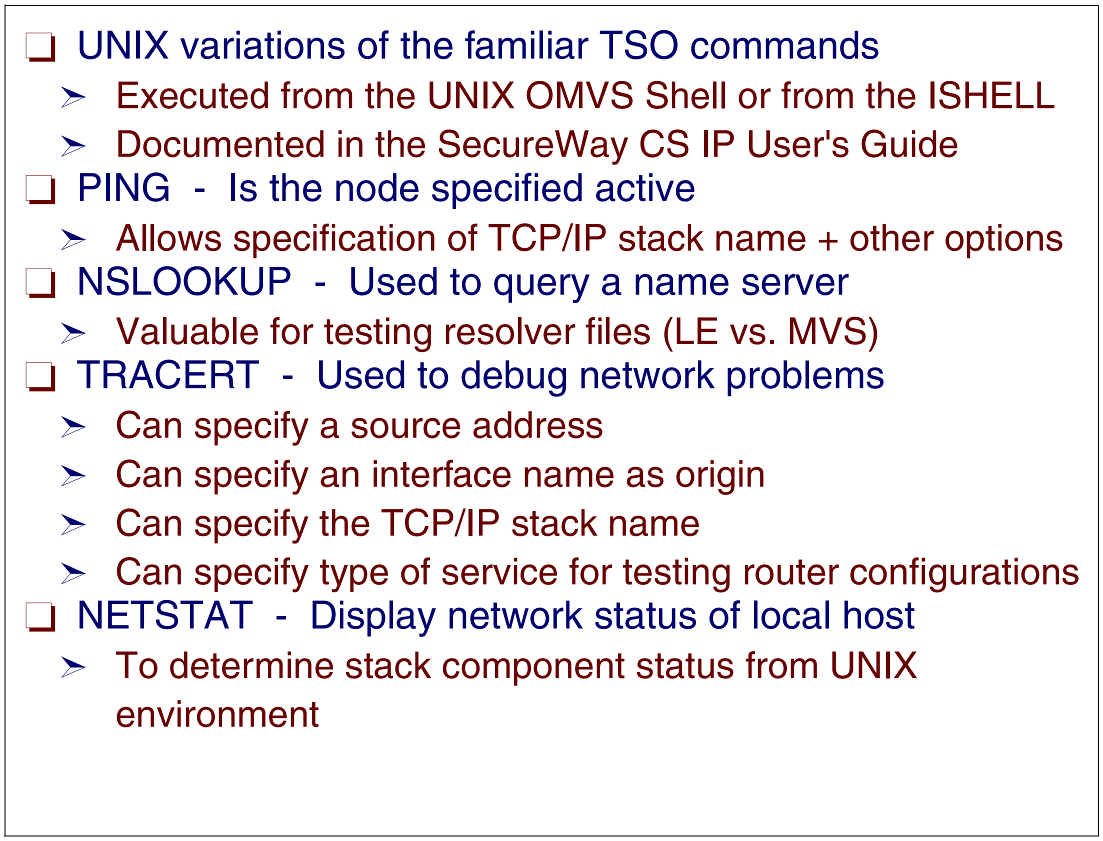
- Copy hlq.SEZAINST(SERVICES) to hlq.ETC.SERVICES. This file specifies the combination of port and services (UPD or TCP) used by TCP/IP.

  To establish a relationship between the servers defined in the /etc/inetd.conf file and specific port numbers in the UNIX System Services environment, ensure that statements have been added to ETC.SERVICES for each of these servers. See the sample ETC.SERVICES installed in the /usr/lpp/tcpip/samples/services directory for how to specify ETC.SERVICE statements for these servers.

  An HFS file /etc/services could also be created instead of this file.

- Copy hlq.SEZAINST(ETCRPC) to hlq.ETC.RPC. This file specifies the port mapper, which used to be called the portmap daemon.

8.15 TCP/IP shell commands



- ❑ UNIX variations of the familiar TSO commands
  - ➢ Executed from the UNIX OMVS Shell or from the ISHELL
  - ➢ Documented in the SecureWay CS IP User's Guide
- ❑ PING - Is the node specified active
  - ➢ Allows specification of TCP/IP stack name + other options
- ❑ NSLOOKUP - Used to query a name server
  - ➢ Valuable for testing resolver files (LE vs. MVS)
- ❑ TRACERT - Used to debug network problems
  - ➢ Can specify a source address
  - ➢ Can specify an interface name as origin
  - ➢ Can specify the TCP/IP stack name
  - ➢ Can specify type of service for testing router configurations
- ❑ NETSTAT - Display network status of local host
  - ➢ To determine stack component status from UNIX environment

Figure 8-15 Commands issued from the shell for TCP/IP

You can use the TSO **PING**, **TRACERTE**, **NETSTAT**, and **NSLOOKUP** commands from the UNIX environment.

The z/OS UNIX **ping** command sends an echo request to a foreign node (remote node) to determine whether the computer is accessible.

When a response to an **ping** command is received, the elapsed time is displayed. The time does not include the time spent communicating between the user and the TCP/IP address space.

Use the **ping** command to determine the accessibility of the foreign node.

Note: **ping** is a synonym for the **oping** command in the z/OS UNIX shell. **ping** command syntax is the same as that for the **oping** command.

The z/OS UNIX **nslookup** command enables you to query any name server to perform the following tasks from the z/OS UNIX environment:

- Identify the location of name servers
- Examine the contents of a name server database
- Establish the accessibility of name servers

9

TCP/IP applications

This chapter provides information on how to customize z/OS UNIX in order to use TCP/IP. It also discusses how to enable remote UNIX logins to the z/OS UNIX System Services shell using rlogin or Telnet, as well as FTP daemon and SYSLOGD daemon.

The chapter begins by describing the remote login procedure and the socket file systems in BPXPRMxx for z/OS UNIX.

Then it provides the details of how to set up the following z/OS UNIX daemons: inetd, rlogin, Telnet, FTPD, and SYSLOGD.

Finally, it explains the search sequence for CS TCP/IP.

9.1 Overview of z/OS UNIX data access

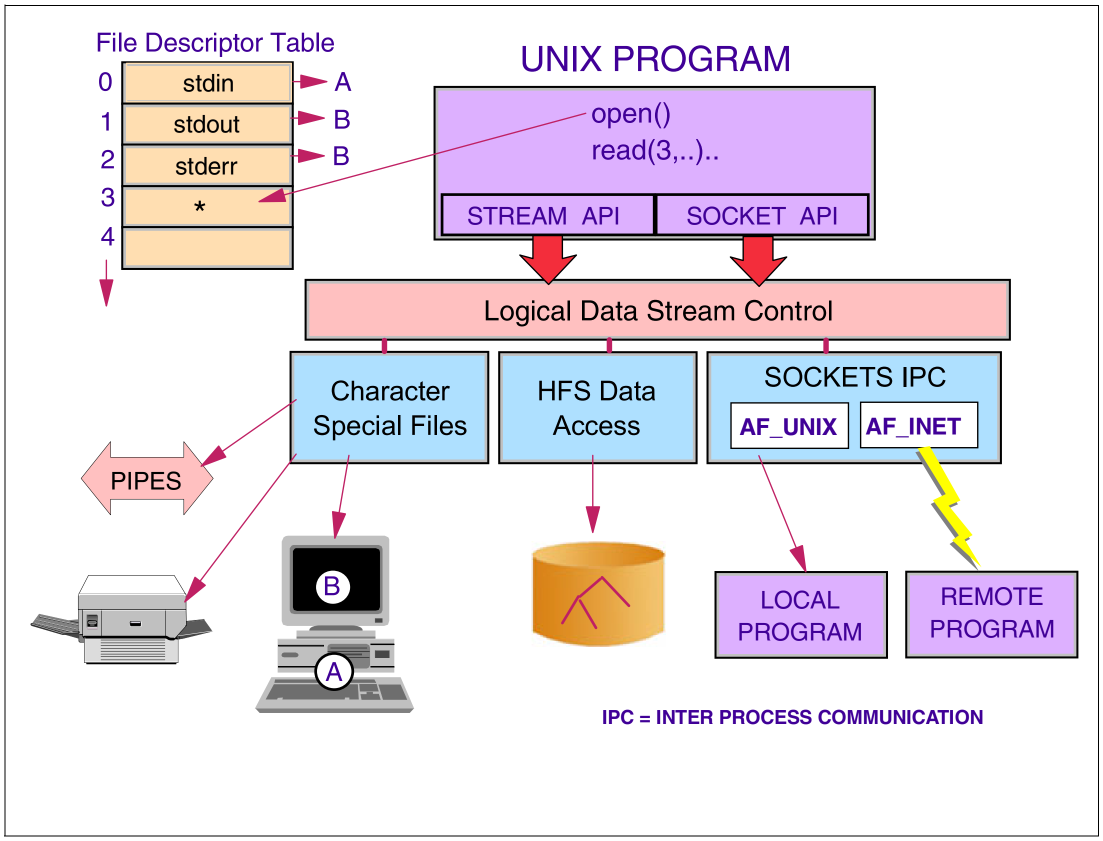


Figure 9-1 User access to z/OS UNIX overview

To do work in a UNIX system, a user requires a program, and data on which the program is to operate. The program is represented by a system process, and the data to manipulate resides in files or streams.

UNIX can use streams to represent many physical data objects:

- *Data files* stored in the UNIX hierarchical file system
- *Special character files* which represent physical devices such as printers and terminals, as well as logical devices such as data pipes
- *Sockets* or message streams used for Inter Process Communication (IPC)

When an application OPENs any stream to process data, the UNIX OPEN routines return a *file descriptor* that is used to identify the stream in future processing calls.

The file descriptor value indexes an entry (file handle or socket handle) stored in a logical table of file handles for files/streams opened by this program.

When an application starts, default file descriptors are automatically opened for:

- STDIN (FD = 0) - Standard Input - user terminal keyboard
- STDOUT (FD = 1) - Standard program output - to user terminal
- STDERR (FD = 2) - Standard error message output - to user terminal

Sockets come in 2 types:

- AF_UNIX sockets (IPC between programs on same host)
- AF_INET sockets (IPC with remote program over network)

Because handling sockets requires more complex functionality than simple data streams, UNIX programs typically use a different Application Programming Interface (API) (set of program calls) to work with sockets.

STDIN, STDOUT, and STDERR should not be new concepts. However, it is important to point out that they get their unique attributes from occupying the first three slots in this file descriptor table (FD=0, FD=1, and FD=2).

For those MVS internals gurus among you, the file descriptor table is similar in concept to the MVS TIOT (Task Input/Output table).

9.2 Sockets

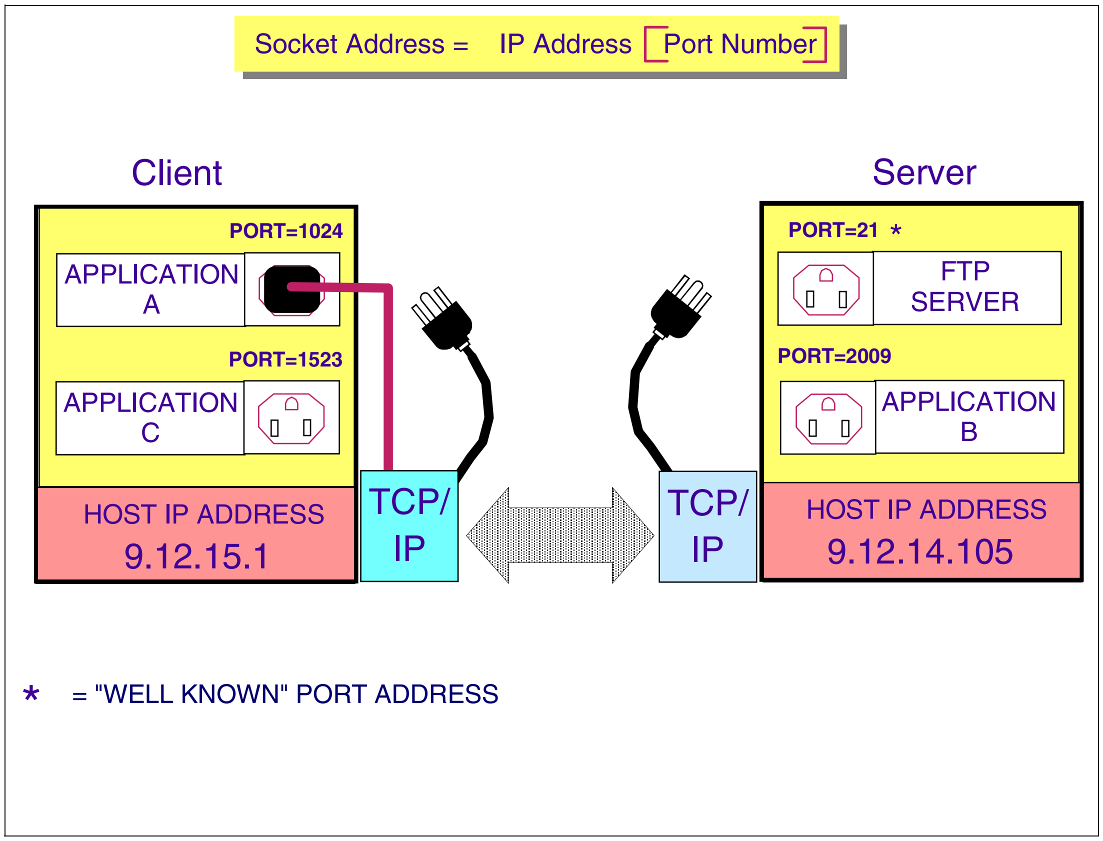


Figure 9-2 z/OS UNIX sockets overview

This should be a refresher topic. It is intended to stress that sockets (at least in the classic UNIX interpretation) are not to be treated any differently than ordinary data files insofar as program logic is concerned. Also, it should be reiterated that sockets are used for local (intrahost) as well as remote (interhost or network) communication between UNIX processes.

Simply position the concept of an API as being the set of C library service calls used to work with a particular resource. Although sockets and HFS files are both streams, the complexity of socket handling requires more powerful syntax and options for handling socket data. This is relevant in a later discussion where we see that z/OS has perhaps 8 different socket APIs!

A powerful design principle was used in UNIX data processing: any type of data manipulated by a process can be represented by a standard file/stream structure. Also, a single set of program calls is used to open streams, read and update data, and close streams, insulating the program from dealing with the physical origin of stream data.

Of course, as with many things in UNIX, this is an idealized picture which fits various commercial UNIX “flavors” to a greater or lesser extent. The currently available UNIX flavors are derived from two main UNIX technologies:

- **Berkeley Software Division (BSD)** - This is UNIX produced by the University of California. It is the strongest influence in UNIX systems produced by DEC Ultrix, SunOS, HP-UX from Hewlett Packard, and, in the early stages, IBM OpenEdition which is now named z/OS UNIX.

- **AT&T** - This is UNIX produced by AT&T, and after that, USL UNIX system laboratories. The current UNIX version is UNIX System VR4. System VR4 is a major influence in IBM AIX, and Apple AUX.

In addition to many other differences, the two flavors of UNIX differ in the way that they see sockets, as follows:

- Berkeley was the original developer of socket protocols, which it connected with TCP/IP network protocols to provide remote process-to-process communication over the network. As a new add-in to the UNIX kernel, and because functionality was considerably more complex in BSD sockets compared to ordinary file handling, BSD sockets use a set of new program calls to manipulate data over sockets. For example, a program would open a file, read or write data to it, and close it. To initialize socket processing, on the other hand, the program issues a socket call, uses **`send`** and **`recv`** commands to transfer data, and then will close the socket.
- The AT&T System V approach is more classical. Both files and sockets are opened, data can be manipulated in both cases using read and write commands, and both are closed to terminate stream usage. Also, AT&T pioneered the idea of the *UNIX-domain* or *local socket*, used to communicate between two UNIX processes in the same system— what MVS would call cross-memory communication. A UNIX domain socket is considerably simpler than a network socket, and can be easily handled with the classic socket approach.

In modern UNIX systems, there has been considerable cross-pollination between the two UNIX flavors, such that, for example, AIX now supports many BSD features, including BSD sockets. Also, BSD sockets provide both local domain and Internet-type socket support. However, when porting the C source code for a new application to a UNIX system, allowances have to be made for the fact that, for example, program calls made to sockets could be written using either the BSD or the System V approach. To prevent code rewrite, system compiler and run-time support must be able to handle both flavors of calls. This principle must apply to z/OS UNIX as well. The original (4.3) approach in OpenEdition was oriented to BSD socket support rather than to System V.

In the classical (System V) approach, all open streams, whether files, terminals, sockets, and so on, are pointed to by descriptors (file handles) in the same file descriptor table. The contents of a socket descriptor will be different from that of a file descriptor in the FDT, but the index of a socket descriptor is an integer number identical to the type of index pointing to a file descriptor for a file. So, for example, a program might have an open data file with file descriptor FD=4, and an initialized socket at socket descriptor SD=5, in consecutive slots of the table. Obviously, the index numbers cannot be allowed to overlap.

9.3 z/OS Communications Server

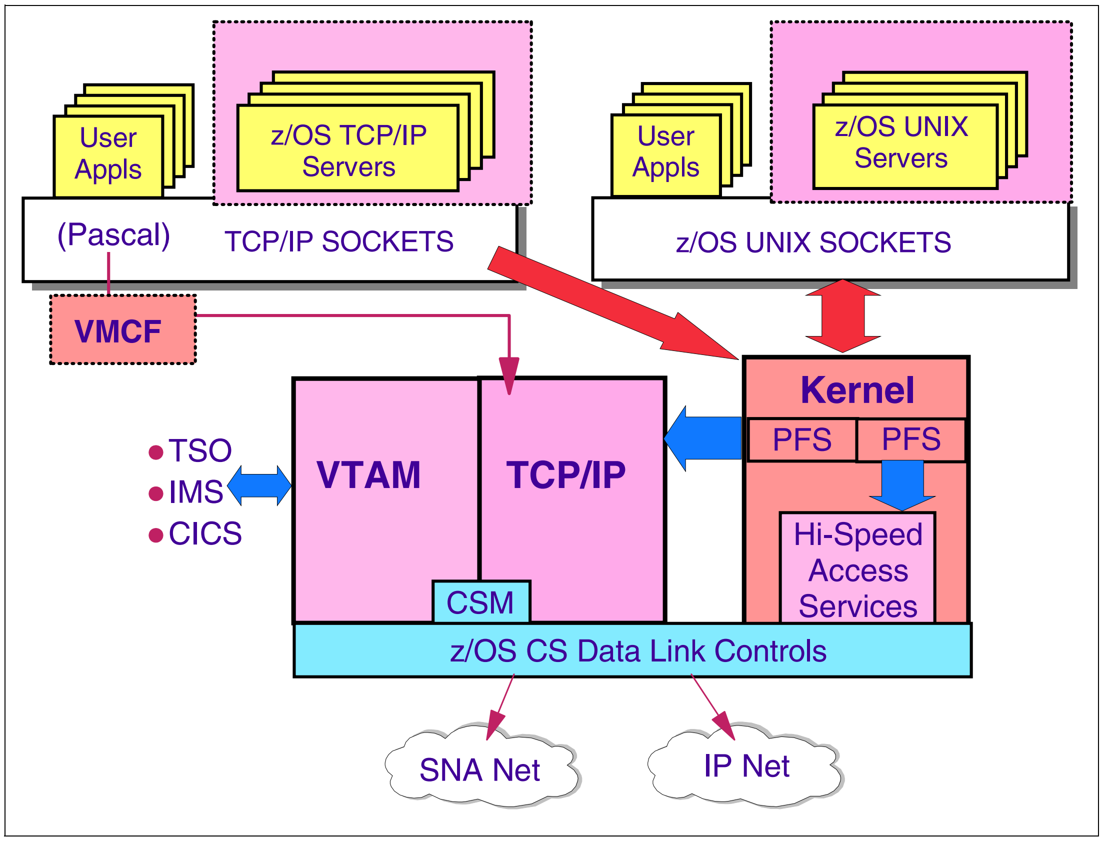


Figure 9-3 Overview of the Communications Server

Since z/OS V2R5 was shipped, the z/OS Communications Server (CS) has been included as a base element. The TCP/IP z/OS component, called z/OS CS, is a reconstructed stack that is able to support both UNIX and non-UNIX socket APIs. It is often called the converged IP stack.

All the TCP/IP socket APIs that supported HPNS are now transparently redirected by run-time support to call the UNIX kernel LFS. The REXX socket API has also been directed to call the kernel. HPNS support is no longer required. The Pascal API, however, still requires the VMCF/IUCV address space to be started, which links the API to the new stack. VMCF and TNF do not respond to commands. This is probably because one or both of the on-restartable versions of VMCF or TNF are still active. To get them to respond to commands, stop all VMCF/TNF users, FORCE ARM VMCF and TNF, then use the EZAZSSI procedure to restart.

The SNA and IP networking stacks have been integrated to a considerable extent. Both stacks use common Data Link Control (DLC) routines to access network hardware, and both types of protocols can flow over the same hardware link. Also, common service routines such as Communications Storage Manager (CSM) exploit use of buffers in common storage for both IP and SNA performance.

9.4 z/OS UNIX sockets with a single stack

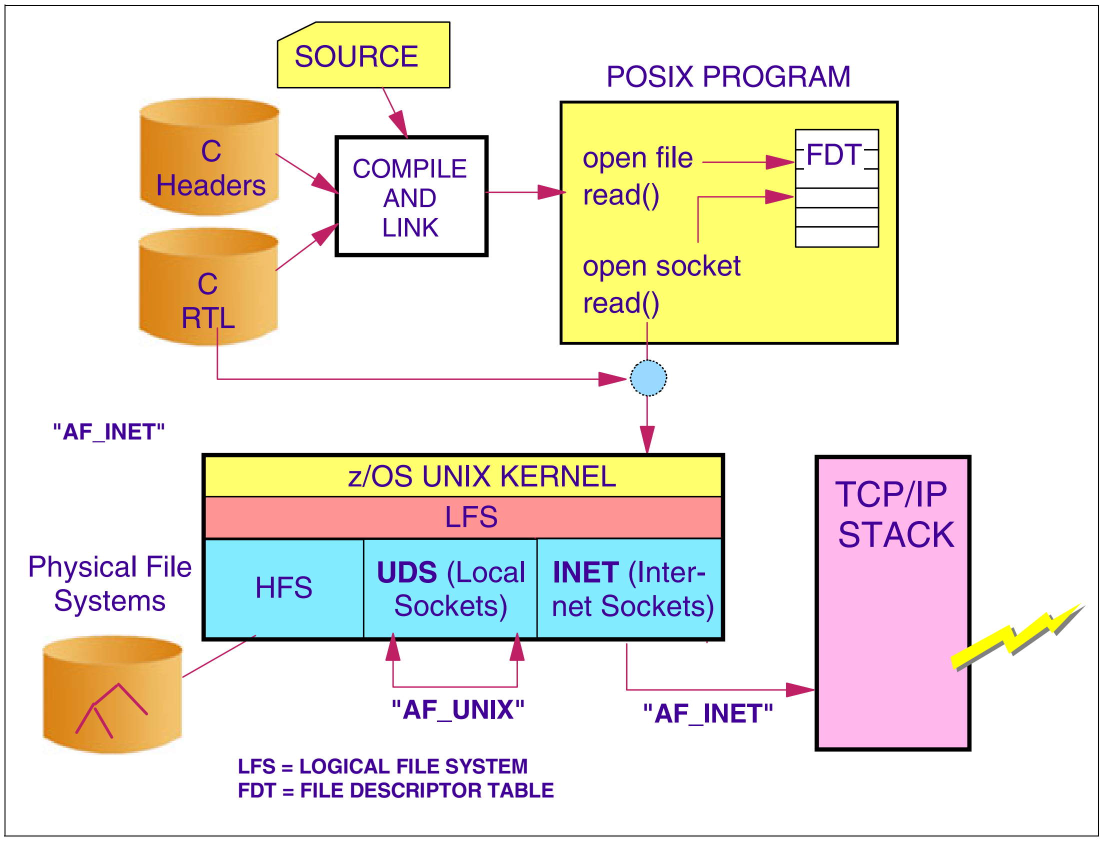


Figure 9-4 Sockets overview with z/OS UNIX

Integrated Socket Support was the UNIX socket API supplied with TCP/IP V3.x. This API makes it transparent to a POSIX process what type of software underpins the C socket calls.

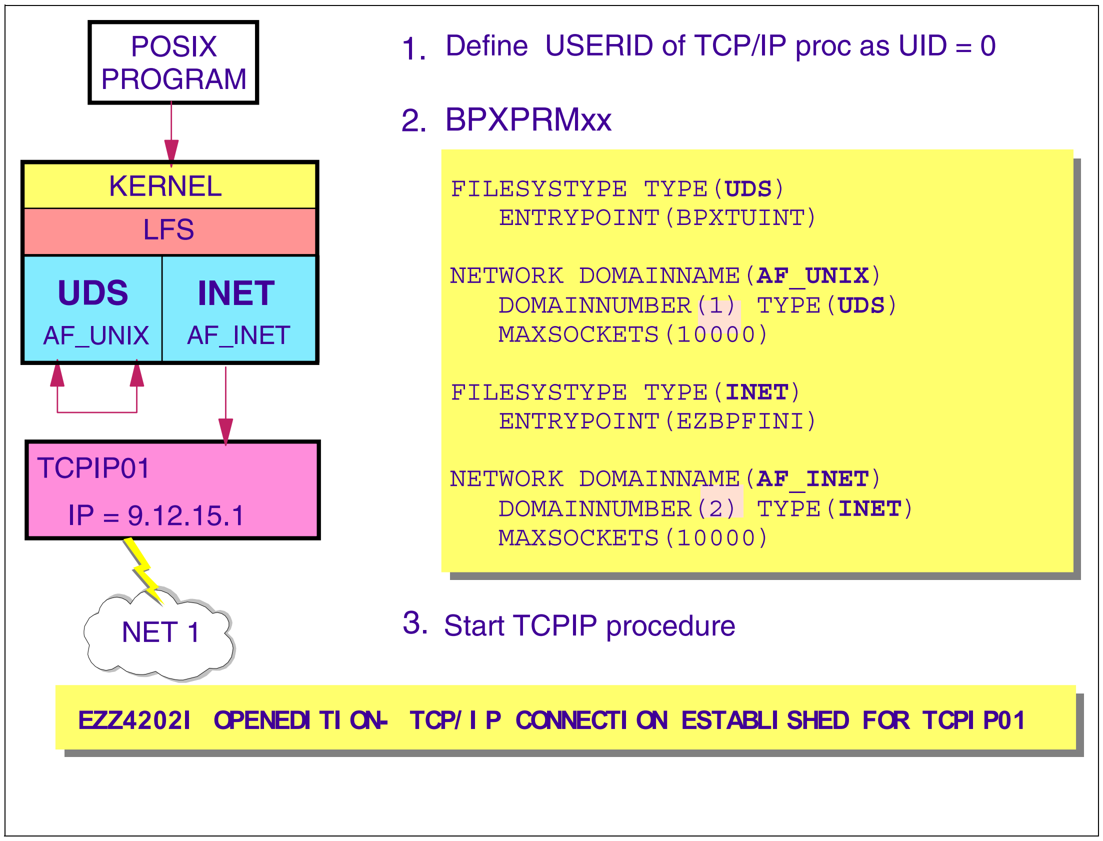
Integrated sockets merged the C runtime library support for TCP/IP (z/OS) and UNIX socket calls. A read() call is supported by one module for files and sockets.

Two Socket Physical File Systems (PFSs) are part of the z/OS UNIX component. One supports AF_UNIX local domain sockets, and the other supports TCP/IP AF_INET network sockets. Both file and socket access are channeled through the standard Logical File System (LFS) to the relevant PFS. This enables z/OS UNIX to use a single pool of numeric IDs to be allocated to both file descriptors and socket descriptors, avoiding any confusion.

Statements to define the PFS components, and a NETWORK statement, are coded in the z/OS UNIX BPXPRMXX PARMLIB member, defining the type of socket support associated with each file system.

A program that uses files and sockets, and uses identical calls (that is, read() and write() and so on) to process both resources can be compiled and linked as a single unit.

9.5 Customizing sockets for a single stack



1. Define USERID of TCP/IP proc as UID = 0
2. BPXPRMxx

```
FILESYSTYPE TYPE(UDS)
   ENTRYPOINT(BPXTUINT)

NETWORK DOMAINNAME(AF_UNIX)
   DOMAINNUMBER(1) TYPE(UDS)
   MAXSOCKETS(10000)

FILESYSTYPE TYPE(INET)
   ENTRYPOINT(EZBPFINI)

NETWORK DOMAINNAME(AF_INET)
   DOMAINNUMBER(2) TYPE(INET)
   MAXSOCKETS(10000)
```

3. Start TCPIP procedure

```
EZZ4202I OPENEDITION-TCP/IP CONNECTION ESTABLISHED FOR TCPIP01
```

Figure 9-5 Customizing the use of z/OS UNIX sockets

This is the first scenario for supporting z/OS UNIX sockets. It assumes a simple environment with only one TCP/IP stack used for network communication.

The user ID owning the TCP/IP address space (as represented by the name of the TCP/IP procedure) must be defined as a z/OS UNIX superuser, by inserting UID=0 in the RACF OMVS segment.

The physical file systems to support socket communication must be defined in the BPXPRMxx member in SYS1.PARMLIB. Two types of statements are required:

- **FILESYSTYPE** defines a z/OS UNIX PFS:
  - **TYPE** - UDS=local domain sockets, INET=networking sockets
  - **ENTRYPOINT** of z/OS UNIX program supporting PFS - code for IP level:
    - BPXTUINT - AF_UNIX (UDS) socket support
    - EZBPFINI - IP stack
    - ISTOEPIT - AnyNet® stack
- **NETWORK** - Describes a group of sockets allocated to a PFS:
  - **DOMAINNAME** - Name of socket file system domain associated with group - local (AF_UNIX) or TCP/IP (AF_INET)
  - **DOMAINNUMBER** - Number matches name - 1=AF_UNIX, 2=AF_INET
  - **MAXSOCKETS** - Max sockets supported - default=100, max=64498
  - **TYPE** - INET or UDS - points to related FILESYSTYPE entry

9.6 Logging in to the z/OS UNIX shell

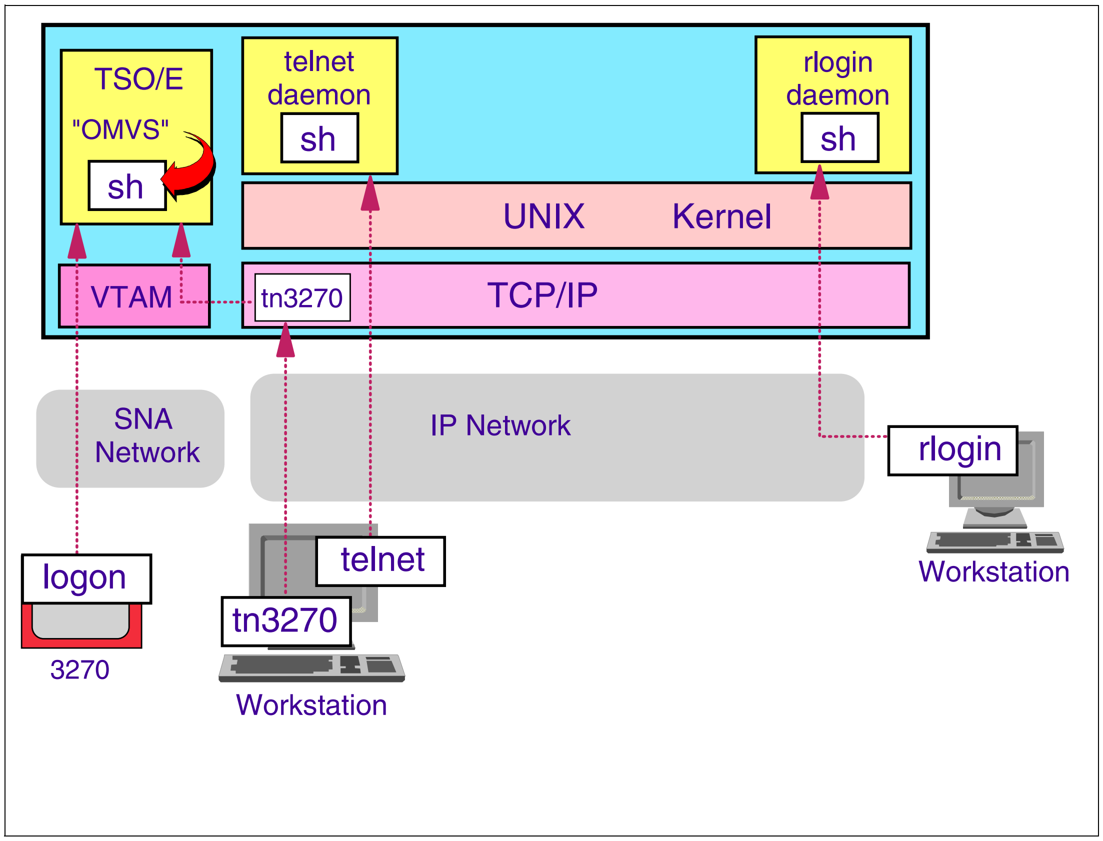


Figure 9-6 Logging in to the z/OS UNIX shell

The following are mechanisms to start a z/OS UNIX shell session:

- Via the TSO **OMVS** command:

  A TSO user, logged in via SNA and VTAM, can issue the OMVS commands directly from the TSO command line.

- A workstation user can use TCP/IP tn3270 protocol to log on to a TSO user ID via IP 3270(E) Telnet server. From TSO, use **OMVS** to access the shell.
- Via **rlogin** done directly from a workstation:

  A UNIX/AIX user can use a standard rlogin client to do remote login to z/OS UNIX. A z/OS UNIX rlogin daemon initiates the shell.

- Via Telnet done directly from a workstation:

  Any platform can use a standard Telnet client to do remote login to z/OS UNIX. A z/OS UNIX Telnet daemon initiates the shell.

- Both direct rlogin/Telnet and CS assisted logins support raw mode.

9.7 Using inetd - master of daemons

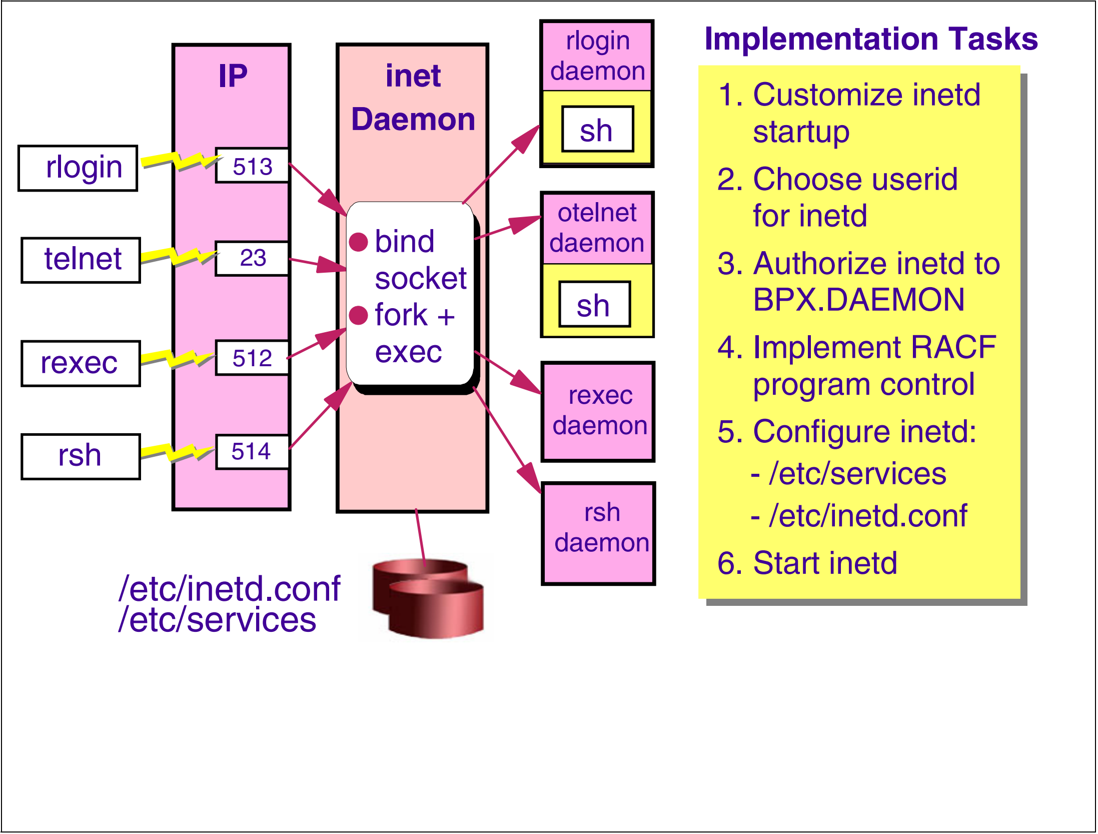


Figure 9-7 Overview of the inetd daemon

The **inetd** daemon, usually known as inetd or InetD, is a master of other daemons that execute in z/OS UNIX. The function of inetd is to listen on certain well-known network ports for a request to run one of a number of daemons. When a request is received, inetd creates a new socket for remote connection, and then fork()s a new address space and uses exec() to start the requested daemon program.

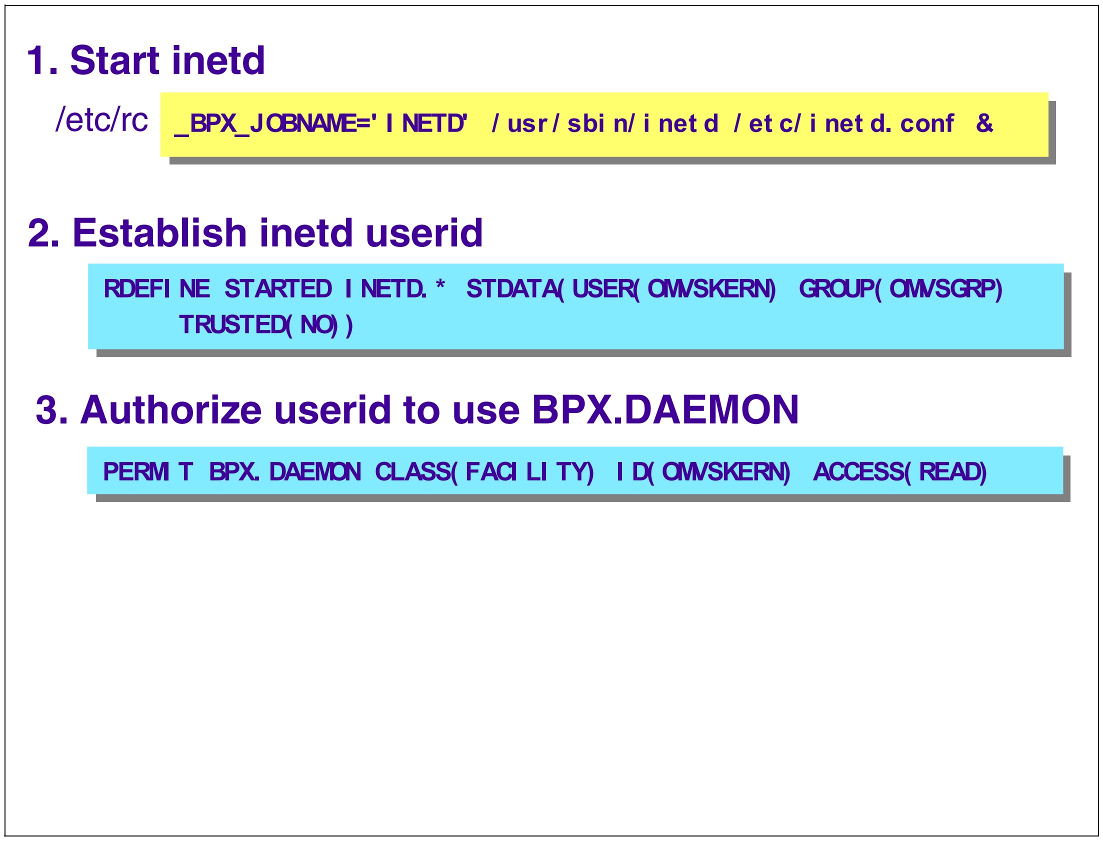
The daemon started by inetd relates to the port where the request arrived. The correlation between port number and daemon is stored in the configuration file /etc/inetd.conf.

The daemons started by inetd include:

- The rlogin daemon starts a shell session for a user rlogin request.
- The Telnet daemon starts a shell session for a user Telnet request.
- The rexec daemon executes a single command on z/OS UNIX requested by a remote user entering a rexec command.
- The rsh daemon starts a shell session and runs a script generated by a remote user entering an rsh command.

Customization is needed to enable inetd to run on your system. You must decide how to start it, and what RACF ID it will execute under. If you have implemented enhanced daemon security with BPX.DAEMON, you must define inetd to the BPX.DAEMON and implement program control. Finally, you have to configure the relationship between the ports that inetd listens on and the daemons to be started.

9.8 Customize inetd



1. Start inetd

/etc/rc

```
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &
```

2. Establish inetd userid

```
RDEFINE STARTED INETD.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP)
      TRUSTED(NO))
```

3. Authorize userid to use BPX.DAEMON

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
```

Figure 9-8 Customizing the inetd daemon

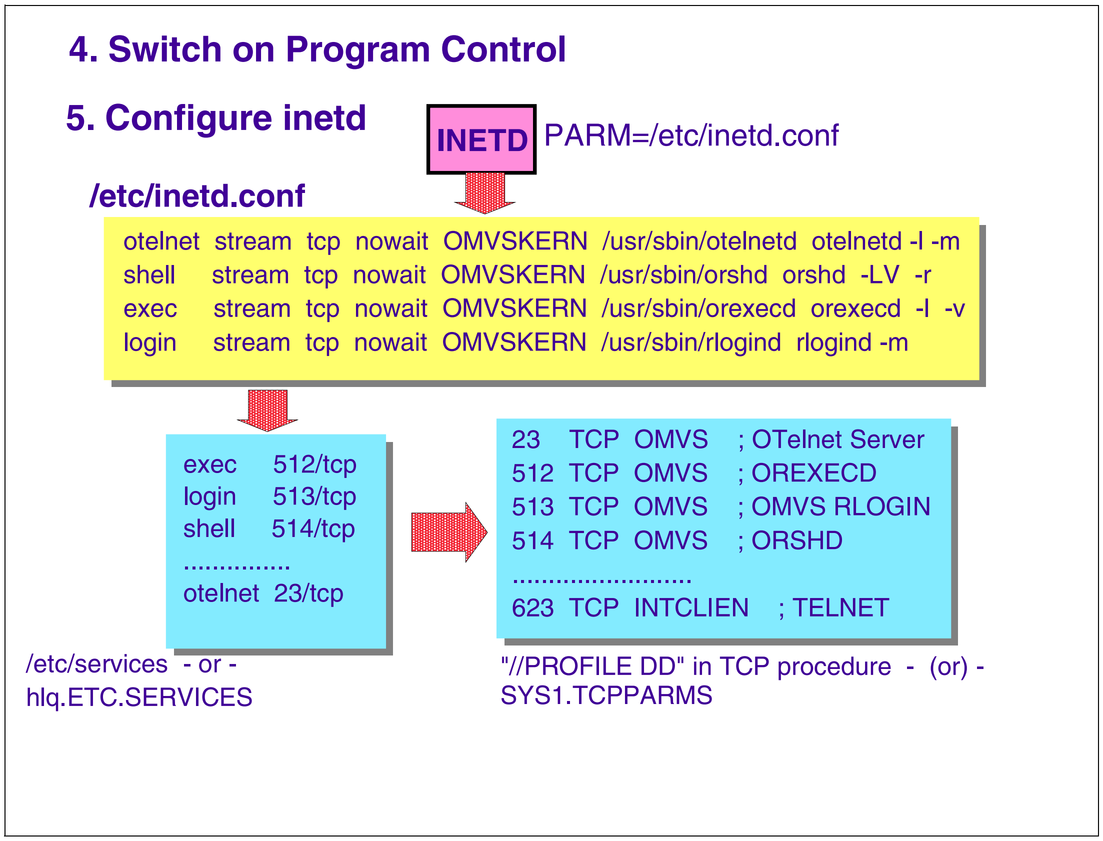
The inetd daemon program can be found in two places. In the HFS, the program file is /usr/sbin/inetd, but IBM has set the sticky bit on. A copy of this program is found in 'SYS1.LINKLIB(INETD)', so this is the program that is used. Start from a line in the initialization script /etc/rc. In this case, use a command similar to the line shown in Figure 9-8.

The next step is to decide which user ID to associate with inetd. It needs to be a superuser (UID=0), and to have minimum access to MVS data sets. How you do this depends on start mode.

When started from /etc/rc, inetd inherits user ID OMVSKERN, which is a superuser. Starting up via /etc/rc you are effectively locked into using the user ID under which the /etc/rc script is running, as inetd is forked from that script. The user ID for /etc/rc is the kernel ID OMVSKERN.

If you have activated the RACF BPX.DAEMON facility, then the INETD user ID must be authorized to this facility.

9.9 Customize inetd (2)



```
otelnet  stream  tcp  nowait  OMVSKERN  /usr/sbin/otelnetd  otelnetd -l -m
shell    stream  tcp  nowait  OMVSKERN  /usr/sbin/orshd  orshd  -LV  -r
exec     stream  tcp  nowait  OMVSKERN  /usr/sbin/orexecd  orexecd  -l  -v
login    stream  tcp  nowait  OMVSKERN  /usr/sbin/rlogind  rlogind -m
```

```
exec     512/tcp
login    513/tcp
shell    514/tcp
...............
otelnet  23/tcp
```

```
23   TCP  OMVS      ; OTelnet Server
512  TCP  OMVS      ; OREXECD
513  TCP  OMVS      ; OMVS RLOGIN
514  TCP  OMVS      ; ORSHD
.........................
623  TCP  INTCLIEN    ; TELNET
```

Figure 9-9 Customizing the inetd daemon

If you have set up the BPX.DAEMON, then you need to make sure that all programs are loaded into the inetd address space. At a minimum, you should protect the following programs:

- SYS1.LINKLIB(INETD).
- CEE.SCEERUN - LE/MVS run time - whole library

There are three configuration files that have to be updated for inetd support:

- The primary file is /etc/inetd.conf, which is the inetd configuration file. There is one entry (line) in this file for each daemon controlled by inetd. The fields are interpreted as follows:
  - Field (1) - Service name - match daemon entry in /etc/services file
  - Field (2) - Daemon socket type - stream or dgram
  - Field (3) - Daemon socket protocol - TCP or UDP
  - Field (4) - Wait_flag - can be wait (single thread server - 1 request at a time) or nowait (multiple requests queued)
  - Field (5) - Login_name - RACF user ID 0nder which daemon will run
  - Field (6) - Server_program - name of daemon program in HFS
  - Field (7) - Server-arguments - first string is jobname for daemon address space, and the rest is the parm string to pass to daemon
- There is a corresponding entry in /etc/services for each daemon in inetd.conf. The entry lists the port where inetd listens for daemon requests.

- The TCPIP PROFILE configuration must list the same ports in the PORT section. This entry identifies the job name authorized to open the socket to this port and the type of socket allowed.

The two TCP/IP files usually exist already; you must make sure that inetd.conf corresponds with the values listed. Also, you may want to change the port number for a daemon.

After all configuration is complete, start inetd.

9.10 Login to a UNIX system

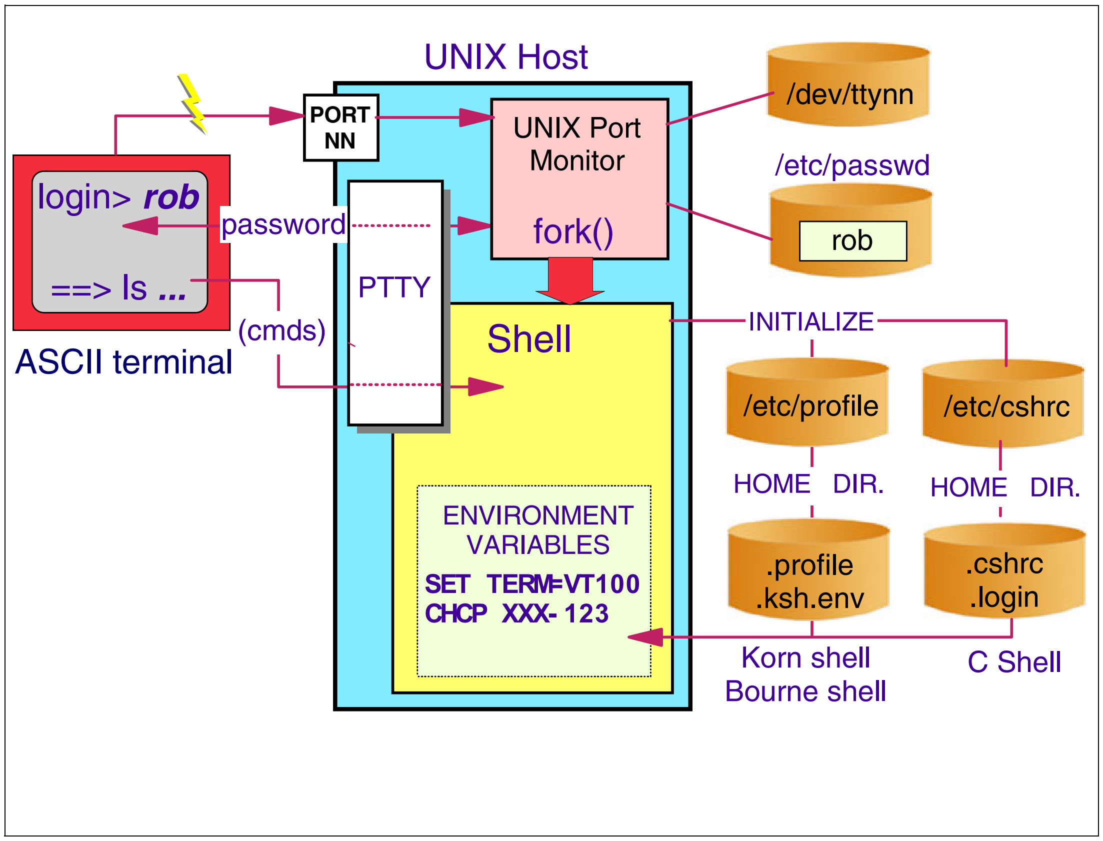


Figure 9-10 Login to a UNIX system on other platforms

Let's look at a real UNIX login approach first for comparison.

When a locally attached ASCII terminal is powered on, an interrupt is generated by the hardware port by which the terminal is connected. This interrupt is detected by the port monitor software, which then retrieves a terminal definition file related to that hardware port. It reads terminal characteristics from this file, initializes a pseudo-TTY terminal interface, and sends the login prompt to the screen.

The user does a login to a locally defined user ID. The port monitor retrieves an account record related to this user from the /etc/passwd file, which includes the user password. The monitor then prompts for and validates the user password.

After validation, the terminal monitor sets the user UID and GID from data in the account file. It then forks a shell process, using the shell program name defined in account data. The PTTY is now the shell terminal interface.

When the shell initializes, it uses the contents of certain system default files, as well as files in the user ID home directory, to initialize the shell environment variables. The files used depend on the type of shell invoked.

In particular, the user may want to set the TERM environment variable in order to define the type of ASCII terminal that the user has. Also, if the user is using a non-US keyboard, they will want to automatically issue the **chcp** (change code page) command to ensure keystrokes are correctly interpreted.

9.11 rlogin to z/OS UNIX services

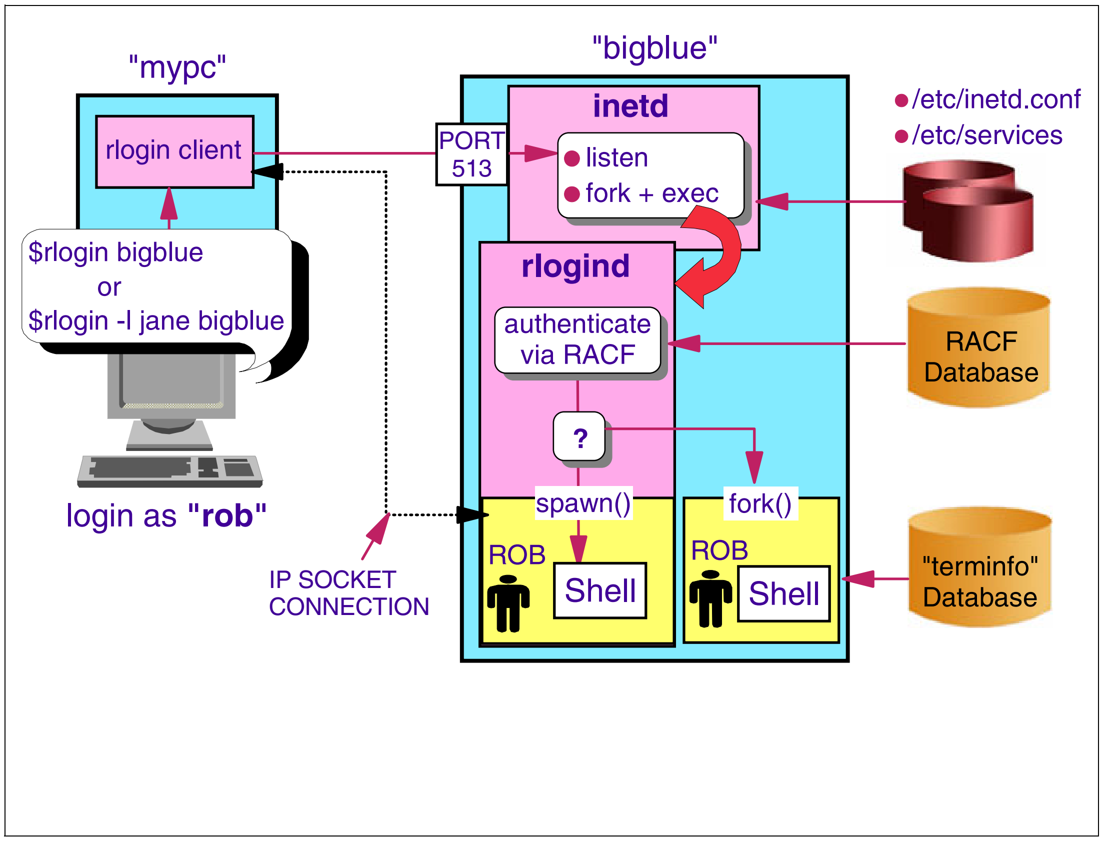


Figure 9-11 rlogin to a z/OS UNIX system

Figure 9-11 illustrates a similar flow through an rlogin request made through TCP/IP to z/OS UNIX. Assume that the user has already done a login to the local host as ***rob***. The user issues `rlogin` from the shell session. The format will depend on the local host. z/OS UNIX accepts an rlogin under the current ID (rob) or the new ID (jane).

The AIX/UNIX rlogin client sends the request to port 513 on the host, monitored by the inetd daemon. inetd forks a new address space and initializes the rlogind server.

The rlogind server uses a z/OS UNIX socket, created by inetd and passed via fork, to communicate with the rlogin client. The server then proceeds to validate the rlogin request, as follows:

- It reads the RACF user profile for the rlogin user ID passed (the current or new user ID). It also reads the contents of the RACF OMVS segment.
- It prompts the remote client for the correct (RACF) password. Note that z/OS UNIX does not support the use of either /etc/equiv.hosts or $HOME/.hosts files defined in HFS to bypass authentication.

If authentication is good, rlogind allocates a standard z/OS UNIX pseudo-tty terminal pair, and then initiates the client shell in one of two ways:

- It creates a child shell process using local_spawn() and the validated user ID.
- It forks a copy of itself in a new address space, uses setuid() and seteuid() commands to set RACF security to a valid user ID, and then runs an exec() shell program.

9.12 Activating z/OS UNIX rlogin daemon

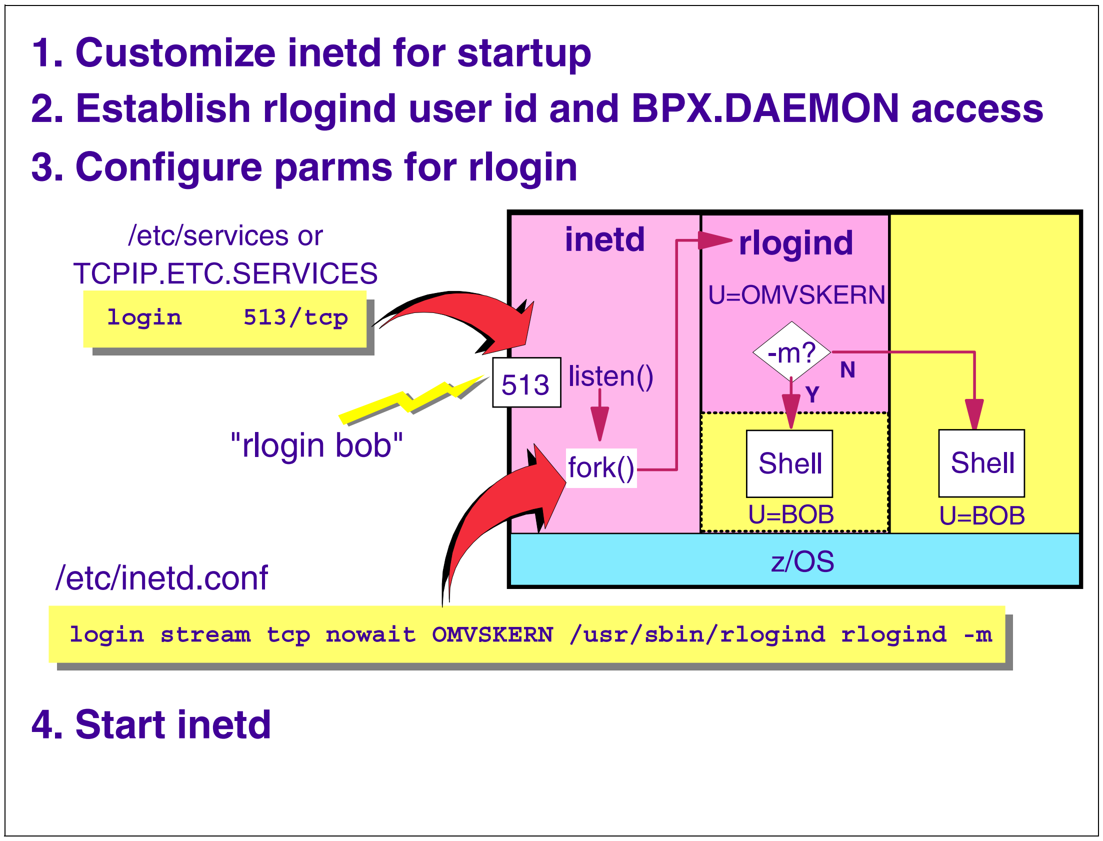

1. Customize inetd for startup
2. Establish rlogind user id and BPX.DAEMON access
3. Configure parms for rlogin
/etc/services or
TCPIP.ETC.SERVICES
login 513/tcp
"rlogin bob"
inetd
513
listen()
fork()
rlogind
U=OMVSKERN
-m?
N
Y
Shell
U=BOB
Shell
U=BOB
z/OS
/etc/inetd.conf
login stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
4. Start inetd

Figure 9-12 Activating the rlogin daemon

Figure 9-12 illustrates the steps to customize z/OS UNIX for the rlogin daemon, as follows:

1. Go through the steps to customize the z/OS UNIX inet daemon INETD, and test that the daemon is able to start.
2. Identify the user ID under which rlogind (the login daemon) will run. The rlogind program as a daemon needs to be a superuser (UID=0), and authorized to access the BPX.DAEMON RACF facility, if used. The kernel user ID is typically used.
3. Configure parms for starting rlogind as follows:
   – Ensure that the TCPIP.ETC.SERVICES file has active entry, as shown in the figure. This assigns port 513 to the rlogin daemon.
   – Update the inetd configuration file, /etc/inetd.conf, to include the entry for the rlogin daemon.
     - **login** - The ID of the entry for rlogin; must match TCPIP.ETC.SERVICES.
     - **stream tcp** - Identifies the daemon socket protocol (this is required).
     - **nowait** - INETD accepts multiple current connections on behalf of rlogind.
     - **OMVSKERN** - The user ID under which the rlogin daemon runs.
     - **/usr/sbin/rlogind** - Pathname of the rlogin daemon program. Sticky bit on means that the system actually fetches SYS1.LINKLIB(RLOGIND).
     - Remaining string = parameters for rlogin daemon (see the following item).
   – Parameters in the rlogind parameter string can include:

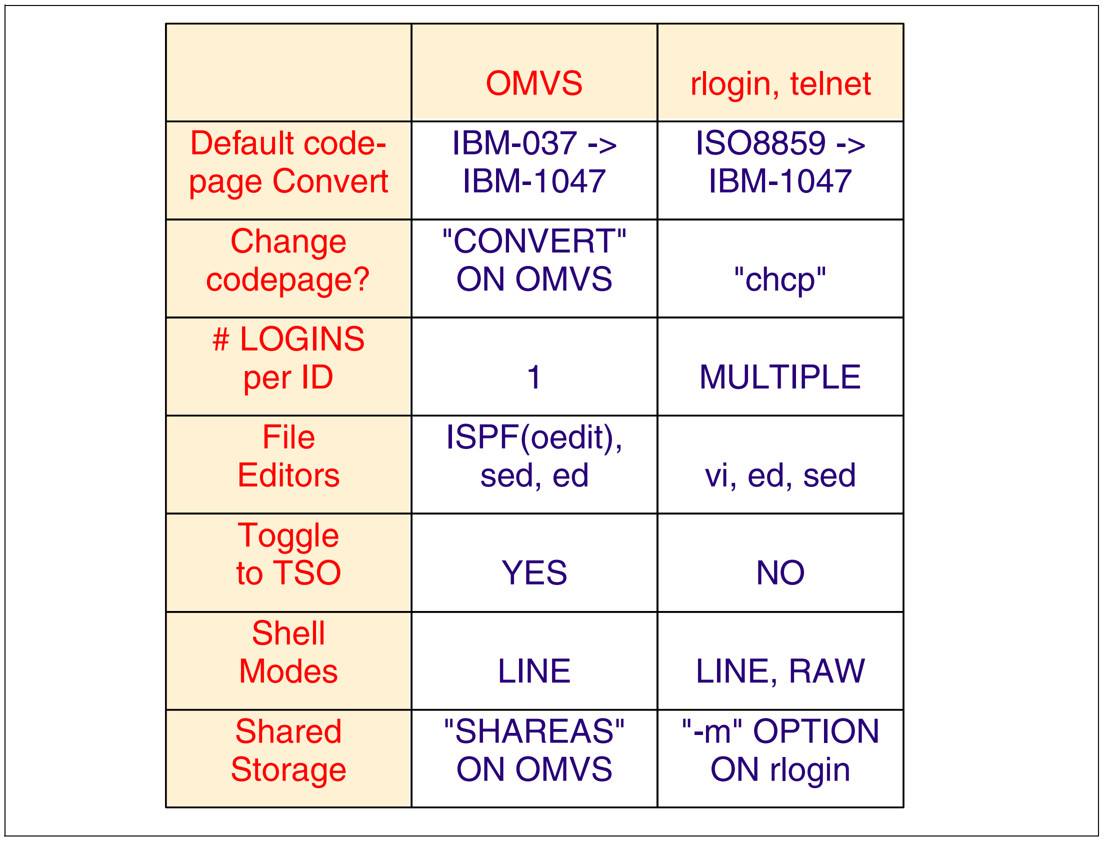
- **rlogind** - (jobname) of server process.
- **-m** - If specified, the shell process shares address space with the rlogin daemon.
- **-d** - Switches on debug - extra messages are written to the system log.

4. To start rlogind support, you need to start the inetd daemon.

Let's walk through the process of doing an rlogin:

- First the inetd daemon starts up, either when the z/OS UNIX kernel is started from the /etc/rc script, or via a `start` command and procedure.
- The inetd daemon reads the configuration file and discovers that it must listen on TCP/IP port 513 for incoming requests for the rlogind daemon (entry login).
- When an incoming request is received on port 513, inetd BINDS a new socket for the request and then forks inetd copy in a new address space.
- inetd copy sets the jobname for the new address space to RLOGIND (from inet.conf parm 7), does setuid to the user ID for rlogon (OMVSKERN), and then does exec () to call the rlogind program. It passes the rest of the argument string from inetd.conf as a parameter.
- The rlogind daemon uses the supplied socket to contact the client and validate the incoming login request. If the client gives a valid ID, rlogind reads the contents of the OMVS segment for the user ID and allocates a PTY/TTY virtual terminal pair for the session.
- Then rlogind tests for the -m parameter. If this is supplied, it runs the shell as a child process in the rlogind address space. Otherwise, rlogind forks a new address space and execs the shell in that address space. In either case, the shell runs under the client user ID.

9.13 Comparing shell login methods



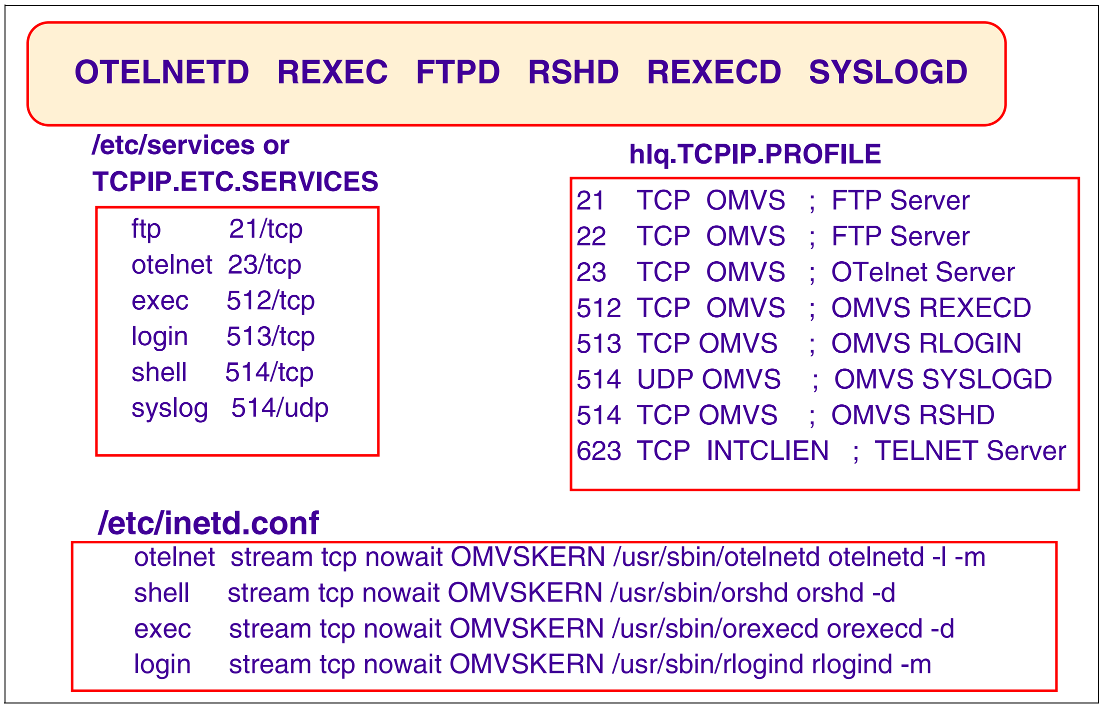
	OMVS	rlogin, telnet
Default code-page Convert	IBM-037 -> IBM-1047	ISO8859 -> IBM-1047
Change codepage?	"CONVERT" ON OMVS	"chcp"
# LOGINS per ID	1	MULTIPLE
File Editors	ISPF(oedit), sed, ed	vi, ed, sed
Toggle to TSO	YES	NO
Shell Modes	LINE	LINE, RAW
Shared Storage	"SHAREAS" ON OMVS	"-m" OPTION ON rlogin

Figure 9-13 Comparison of the logins to the shell

Figure 9-13 compares and contrasts the different methods of accessing the shell, as follows:

- **Default code page translation** - The OMVS shell interface translates between IBM-037 and IBM-1047. However, it does not handle C square brackets or the accent correctly. All other logins translate between ISO-8859 ASCII and IBM-1047.
- **Changing code page** - For OMVS, use the `CONVERT` option on the **`OMVS`** command. For all other methods, use the **`chcp`** shell command after login.
- **Number of concurrent logins** - OMVS users can only start one shell session at a time. All asynchronous users can log in to multiple shell sessions concurrently.
- **File editors** - In OMVS, the ISPF full screen editor can be used via the **`oedit`** command. No asynchronous modes can use ISPF, but they can use the **vi** full screen editor. **ed** and **sed** line editors are commonly available.
- **Shell mode** - OMVS can only work in line (canonical) mode. Asynchronous login supports line mode or *raw* (non-canonical) mode.
- **Toggle to TSO?** - Only OMVS shell provides this option.
- **Shared address space** - Use a single address space to support both terminal I/O and shell processes. For OMVS, specify the `SHAREAS` parameter on the **`OMVS`** command. For rlogin, use the `-m` option on the **`start`** command for the lm daemon.

9.14 Define TCP/IP daemons



OTELNETD REXEC FTPD RSHD REXECD SYSLOGD

/etc/services or TCPIP.ETC.SERVICES

```
ftp       21/tcp
otelnet   23/tcp
exec      512/tcp
login     513/tcp
shell     514/tcp
syslog    514/udp
```

hlq.TCPIP.PROFILE

```
21   TCP  OMVS     ; FTP Server
22   TCP  OMVS     ; FTP Server
23   TCP  OMVS     ; OTelnet Server
512  TCP  OMVS     ; OMVS REXECD
513  TCP  OMVS     ; OMVS RLOGIN
514  UDP  OMVS     ; OMVS SYSLOGD
514  TCP  OMVS     ; OMVS RSHD
623  TCP  INTCLIEN ; TELNET Server
```

/etc/inetd.conf

```
otelnet  stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m
shell    stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -d
exec     stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -d
login    stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
```

Figure 9-14 Defining the TCP/IP daemons

The TCP/IP z/OS UNIX Application feature provides several other TCP/IP functions that you might want to configure, as follows:

TELNET Allows remote users to log in to z/OS using a Telnet client. The z/OS UNIX Telnet server is started for each user by the INETD listener program.

REXEC Remote execution client and server support the sending and receiving of a command.

FTP File transfer program supports transfer into and out of the HFS.

RSH Provides remote execution facilities with authentication based on privileged port numbers, user IDs, and passwords.

SYSLOGD Supplies the logging functions for programs that execute in the z/OS UNIX environment.

Ports need to be assigned to the functions that you choose to configure. The hlq.TCPIP.PROFILE data set has an entry for each function and its port and protocol. If you will be configuring both the z/OS UNIX version and the standard TCP/IP version, you will need to decide which one will use the well-known port assignment.

The TCP/IP resolver function also needs to have the port assignments. These can reside in either the TCPIP.ETC.SERVICES data set or the /etc/services file.

Each daemon then has its own configuration information. The inetd program comes with z/OS UNIX and is the listener program for several of the TCP/IP daemons. The commands inetd will use to initiate each program are put in the /etc/inetd.conf file.

The SYSLOG and FTP daemons have their own configuration files, /etc/syslog.conf and /etc/ftpd.data respectively, and each requires a startup procedure.

9.15 syslogd daemon

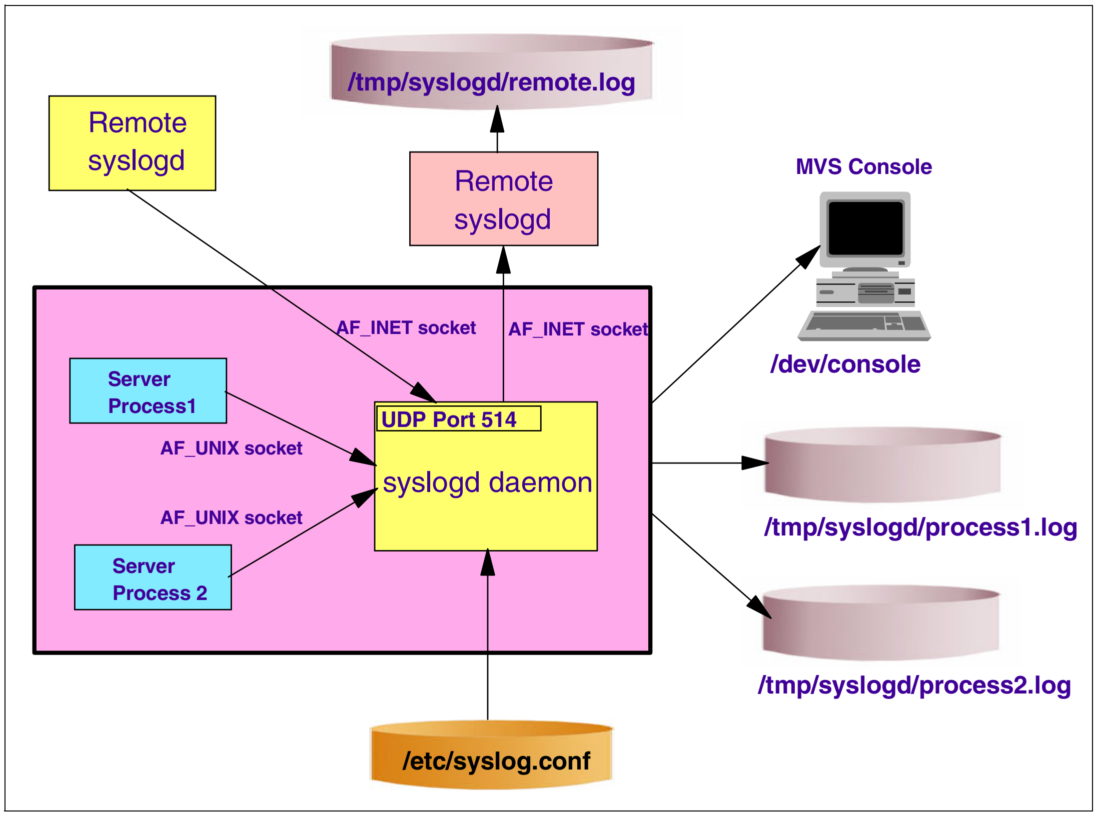


Figure 9-15 Overview of the syslogd daemon

Syslog daemon (syslogd) is a server process that must be started as one the first processes in your z/OS UNIX environment. CS for z/OS server applications and components use syslogd for logging purposes and can also send trace information to syslogd. Servers on the local system use AF_UNIX sockets to communicate with syslogd; remote servers use the AF_INET socket.

The syslogd daemon reads and logs system messages to the MVS console, log files, other machines, or users, as specified by the configuration file /etc/syslog.conf. A sample is provided in /usr/lpp/tcpip/samples/syslog.conf.

If the syslog daemon is not started, the application log may appear on the MVS console.

The syslog daemon must have a user ID; for example, SYSLOGD defined in RACF with UID=0. The syslogd daemon uses the following files:

/dev/console	Operator console
/etc/syslog.pid	Location of the process ID
/etc/syslog.conf	Default configuration file
/dev/log	Default log path for z/OS UNIX datagram socket
/usr/sbin/syslog	Syslog server

syslogd can only be started by a superuser. It can be terminated using the SIGTERM signal.

If you want syslogd to receive log data from or send log data to remote syslogd servers, reserve UDP port 514 for the syslogd job in your PROFILE.TCP/IP data set and enter the syslog service for UDP port 514 in the services file or data set (for example, /etc/services).

9.16 FTPD daemon

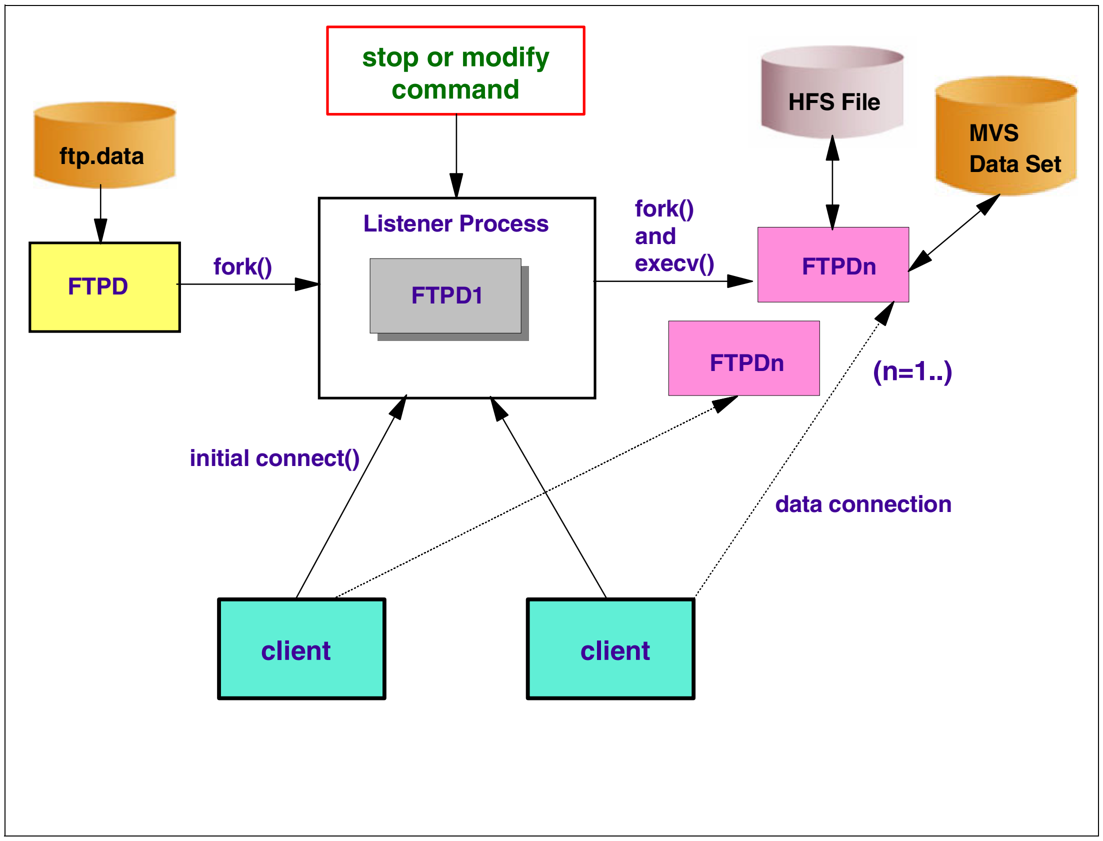


Figure 9-16 Overview of the FTPD daemon

File Transfer Protocol (FTP) is used to transfer files between TCP/IP hosts. The FTP client is the TCP/IP host that initiates the FTP session, while the FTP server is the TCP/IP host to which the client connects.

The FTP server uses two different ports and manages two TCP connections as follows:

- Port 21 is used to control the connection (user ID and password).
- Port 20 is used for actual data transfer based on the FTP client's requests.

The FTP server in z/OS IP consists of the daemon (the listener) or **ftpd** and server address space (or processes). The daemon performs initialization, listens for new connections and starts a separate server address space for each connection.

When a new client FTP connects to the FTPD daemon process, ftpd forks an FTP server process; thus, a new jobname is generated by z/OS UNIX System Services.

9.17 z/OS IP search order for FTP

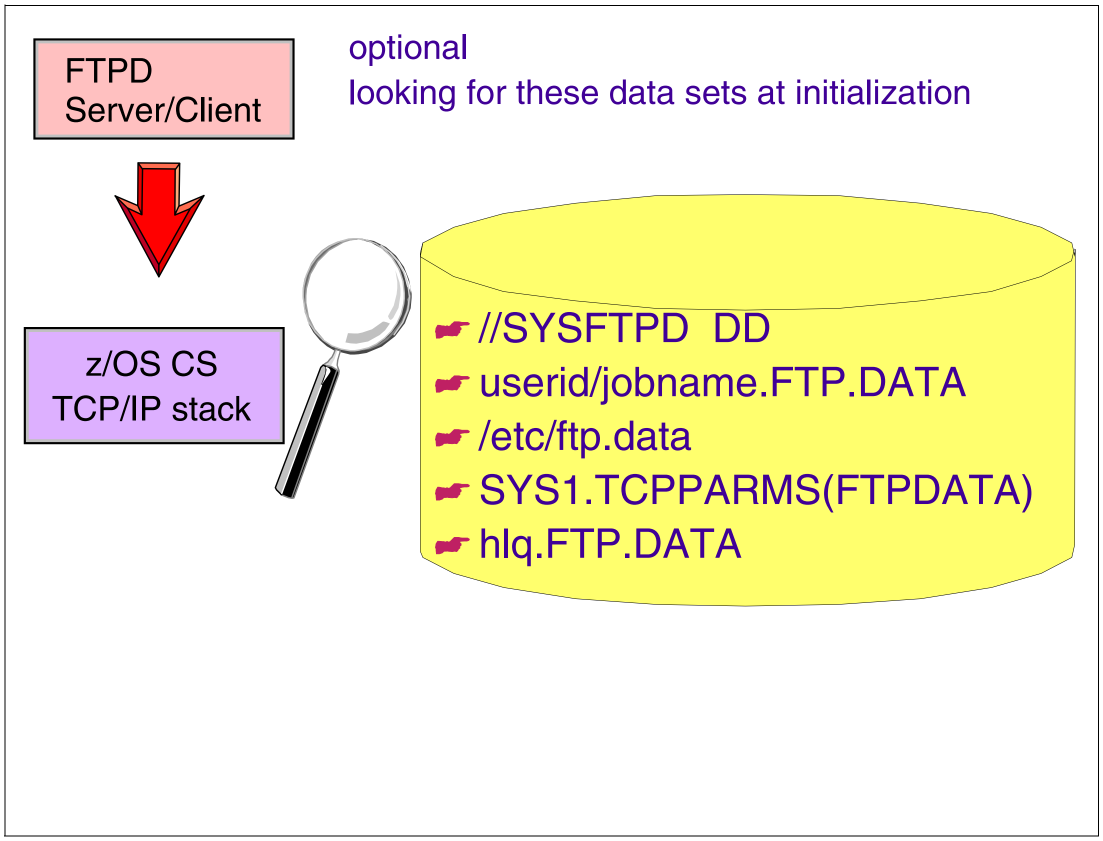


Figure 9-17 Search order for the FTP daemon

FTP.DATA is used to override the default FTP client and server parameters for the FTP server.

You may not need to specify the FTP.DATA data set if the default parameters are used.

A sample is provided in hlq.SEZAINST(FTPDATA) for the client and hlq.SEZAINST(FTPSDATA) for the server.

When an FTPD daemon or started task is started, it searches the FTP.DATA file in the following order:

- //SYSFTPD DD in FTPD started task procedure
- userid/jobname.FTP.DATA
- /etc/ftp.data
- SYS1.TCPPARMS(FTPDATA)
- hlq.FTP.DATA

The search stops if one of these data sets is found.

9.18 z/OS IP search order for /etc/services

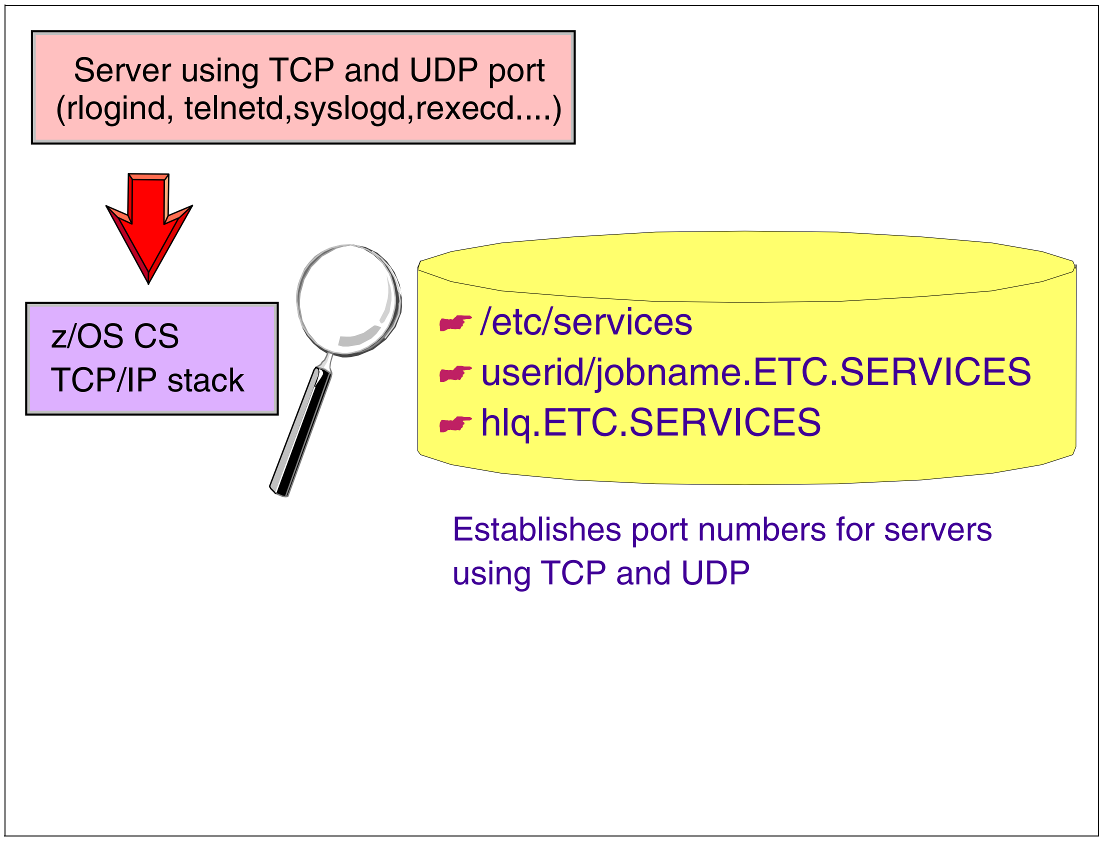

Server using TCP and UDP port
(rlogind, telnetd,syslogd,rexecd....)
z/OS CS
TCP/IP stack
/etc/services
userid/jobname.ETC.SERVICES
hlq.ETC.SERVICES
Establishes port numbers for servers
using TCP and UDP

Figure 9-18 Search order for /etc/services

The ETC.SERVICES data set is used to establish port numbers for UNIX application servers using TCP and UDP. This file or data set is required for any daemon or application that needs the use of a specific port.

Standard applications, like telnet or FTP, are assigned port numbers in the well-known port number range. You can assign port numbers to your own server applications by adding entries to the /etc/services file.

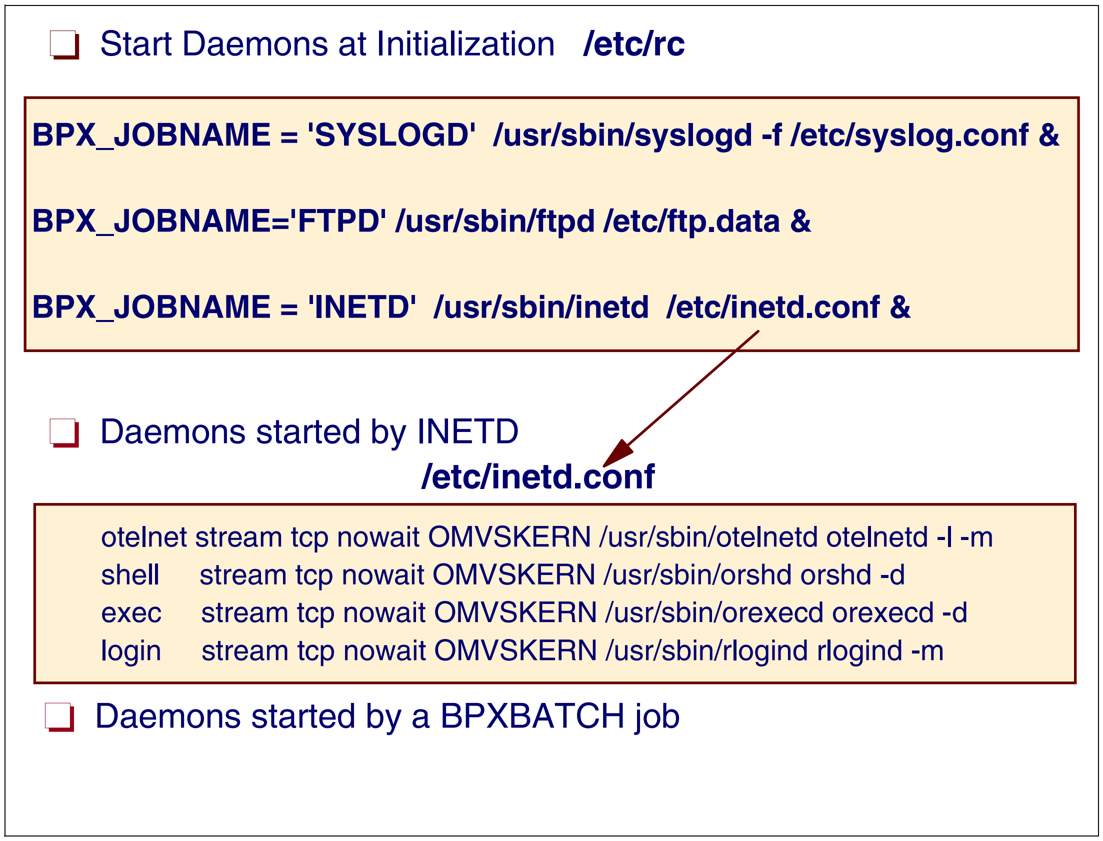
For example, rlogind listens on 513/TCP and telnetd listens on port 23/TCP, while syslogd listens on port 514/UDP. This specification is provided in the ETC.SERVICES data set.

When TCP/IP and the daemons start, they look for the ETC.SERVICE file or data set in the following order:

- /etc/services (HFS file)
- userid/jobname.ETC.SERVICES
- hlq.ETC.SERVICES

The search stops if one of these data sets is found.

9.19 Start the TCP/IP daemons



❑ Start Daemons at Initialization **/etc/rc**

```
BPX_JOBNAME = 'SYSLOGD'  /usr/sbin/syslogd -f /etc/syslog.conf &

BPX_JOBNAME='FTPD' /usr/sbin/ftpd /etc/ftp.data &

BPX_JOBNAME = 'INETD'  /usr/sbin/inetd  /etc/inetd.conf &
```

❑ Daemons started by INETD

/etc/inetd.conf

```
otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m
shell    stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -d
exec     stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -d
login    stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
```

❑ Daemons started by a BPXBATCH job

Figure 9-19 Starting the TCP/IP daemons

After the configuration files have been completed, the daemons need to be started before any remote requests can be processed.

The /etc/rc script is a good place to put the start command, as Figure 9-19 shows. In this case, the daemons will be started during the initialization processing for z/OS UNIX. The _BPX_JOBNAME environment variable will give the daemon an MVS jobname.

Since inetd is responsible for starting the other daemons (Telnet, rlogin, remote shell and remote execution), start commands for them are in inetd's configuration file.

In case any of these daemons fail, you should have other procedures created to restart them since /etc/rc is only used at z/OS UNIX initialization. You could use shell scripts or MVS procedures for this.

10

z/OS UNIX parmlib members

This chapter describes the z/OS UNIX parmlib members. It explains the differences between the BPXPRMxx and BPXPRMFS members and defines the UNIX System Service parameters in each member.

10.1 BPXPRMxx parmlib member

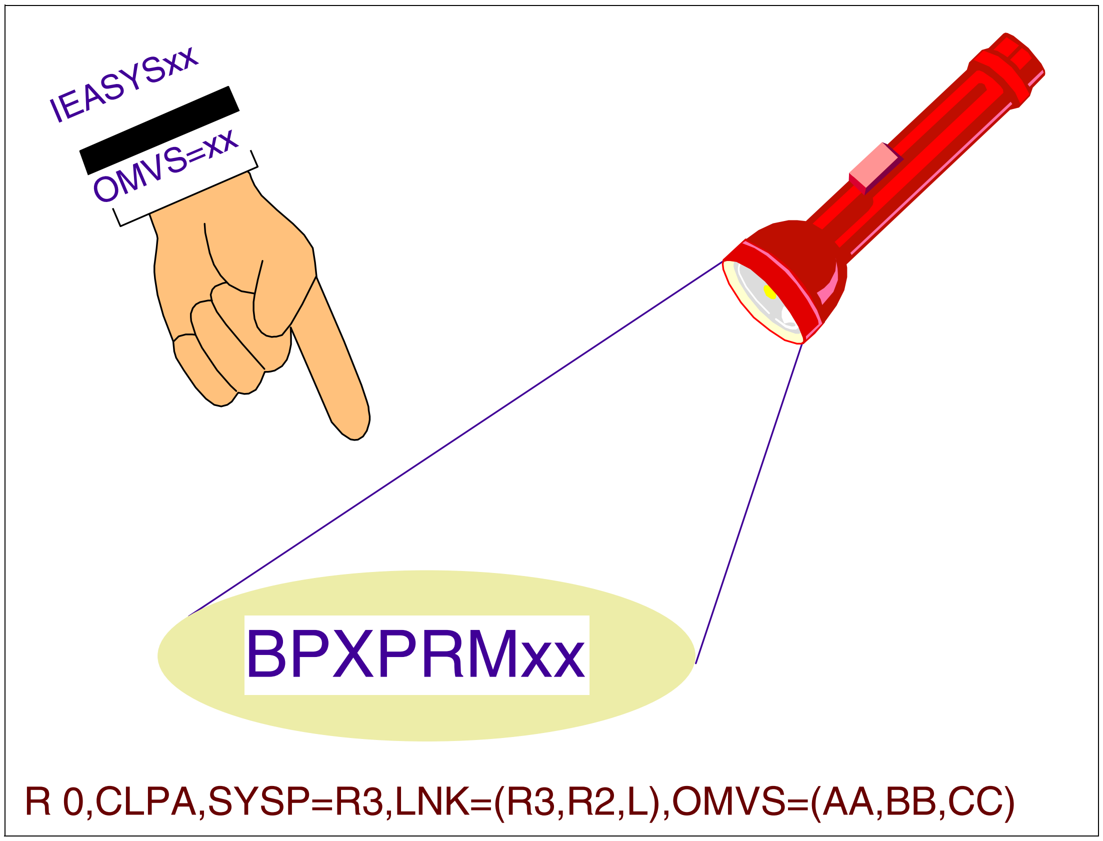


Figure 10-1 z/OS UNIX BPXPRMxx parmlib member

BPXPRMxx contains the parameters that control the UNIX System Services (z/OS UNIX) environment. Beginning with OS/390 V2R3, z/OS UNIX services is started during initialization.

To specify which BPXPRMxx Parmlib member to start with, the operator can include OMVS=xx in the reply to the IPL message or can include OMVS=xx in the IEASYSxx parmlib member. The two alphanumeric characters, represented by xx, are appended to BPXPRM to form the name of the BPXPRMxx Parmlib member.

You can use multiple parmlib members to start OMVS. This is shown by the following reply to the IPL message:

```
R 0,CLPA,SYSP=R3,LNK=(R3,R2,L),OMVS=(AA,BB,CC)
```

The parmlib member BPXPRMCC would be processed first, followed by and overridden by BPXPRMBB, followed by and overridden by BPXPRMAA. This means that any parameter in BPXPRMAA has precedence over the same parameter in BPXPMRBB and BPXPRMCC.

You can also specify multiple OMVS parmlib members in IEASYSxx. For example:

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

To modify BPXPRMxx parmlib settings without re-IPLing, you can use the **`SETOMVS`** operator command, or you can dynamically change the BPXPRMxx parmlib members that are in effect by using the **`SET OMVS`** operator command.

10.2 BPXPRMFS parmlib member

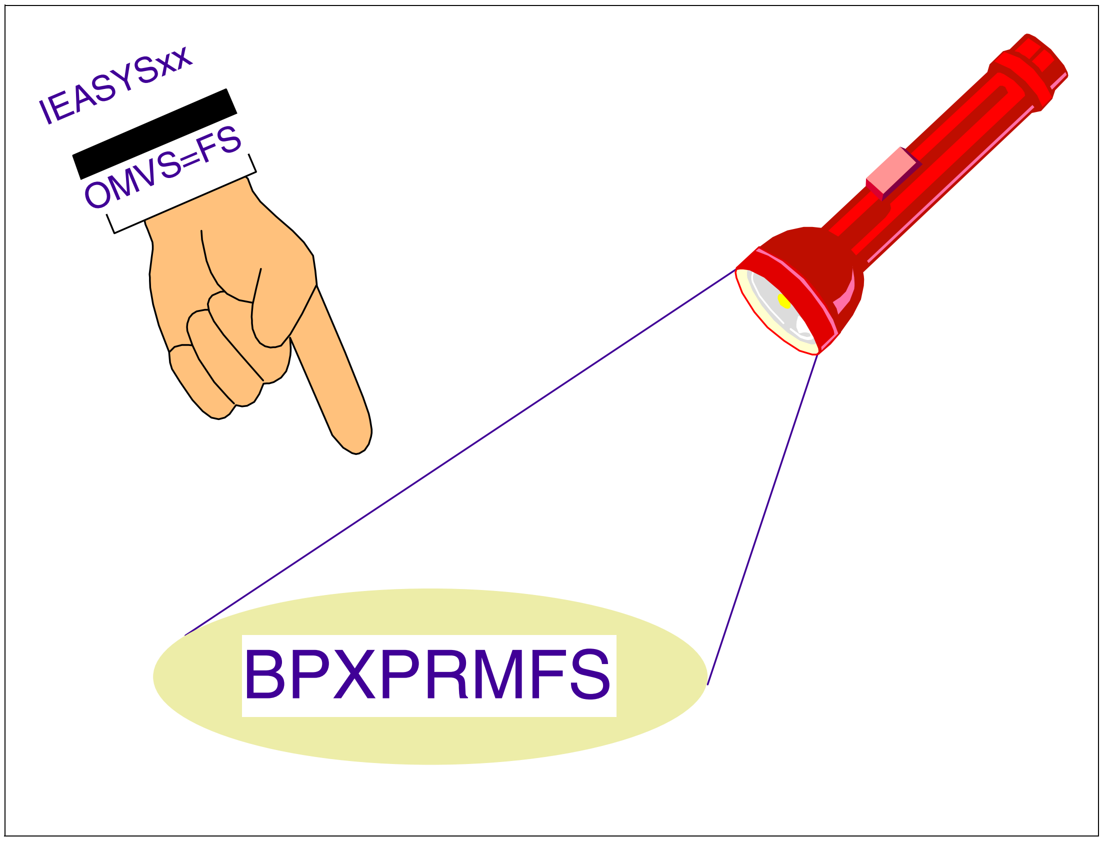


Figure 10-2 BPXPRMFS parmlib member

Starting with the OS/390 V2R6 ServerPac, IBM provided a UNIX System Services parmlib member for HFS and socket definitions:

BPXPRMFS

The BPXPRMFS member reflects the restored file system. The BPXPRMFS parmlib member is shipped in CPAC.PARMLIB and contains:

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

The ALLOCDS job of the ServerPac allocates HFS data sets as you defined in the installation variables option of the installation dialog. The RESTFS job MOUNTs the new ROOT file system and creates mount points for and MOUNTs the additional file systems. The BPXPRMFS member of CPAC.PARMLIB is updated to reflect the file system structure.

It is up to the customer whether to use this additional parmlib member or not. Because you have to IPL when one of these parameters is changed, you should provide the HFS and socket information in a separate member.

10.3 BPXPRMxx control keywords

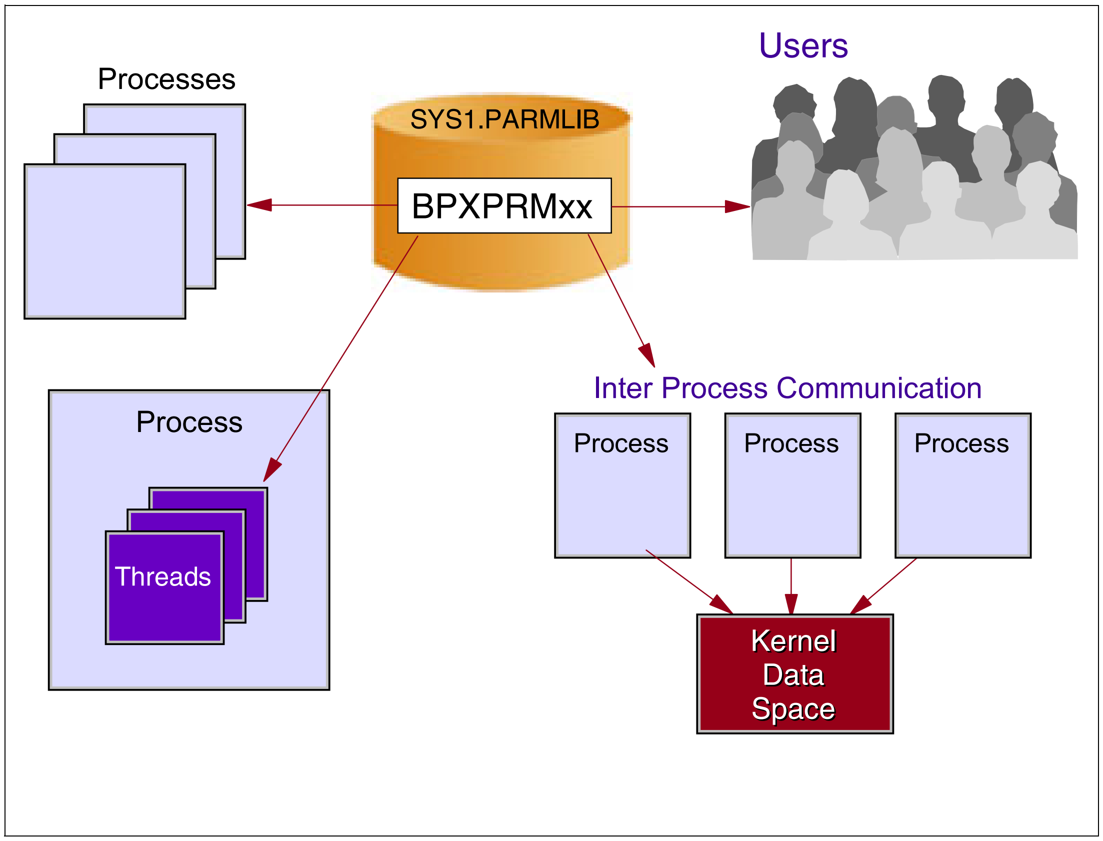


Figure 10-3 BPXPRMxx keywords that control z/OS UNIX processing

The BPXPRMxx parmlib member contains the parameters that control z/OS UNIX processing and the file system. The MVS system uses these values when initializing UNIX System Services.

A sample is shipped in SYS1.SAMPLIB(BPXPRMxx). Copy and rename (if necessary) this member to PARMLIB. Customize this parmlib member according to your installation requirements.

IBM recommends that you have two BPXPRMxx members, one that specifies system limits and one that specifies file system setup. This makes it easier to migrate from one release to another, especially when using the ServerPac method of installation.

You can also define system symbols in the IEASYSxx parmlib member to enable common BPXPRMxx members to be shared by systems. Symbols like system name (&SYSNAME.) can be used in the BPXPRMxx member, specifically when referring to HFS data set names, in order to have different HFS data sets mounted as the root on each system in the sysplex.

10.4 BPXPRMxx parmlib member

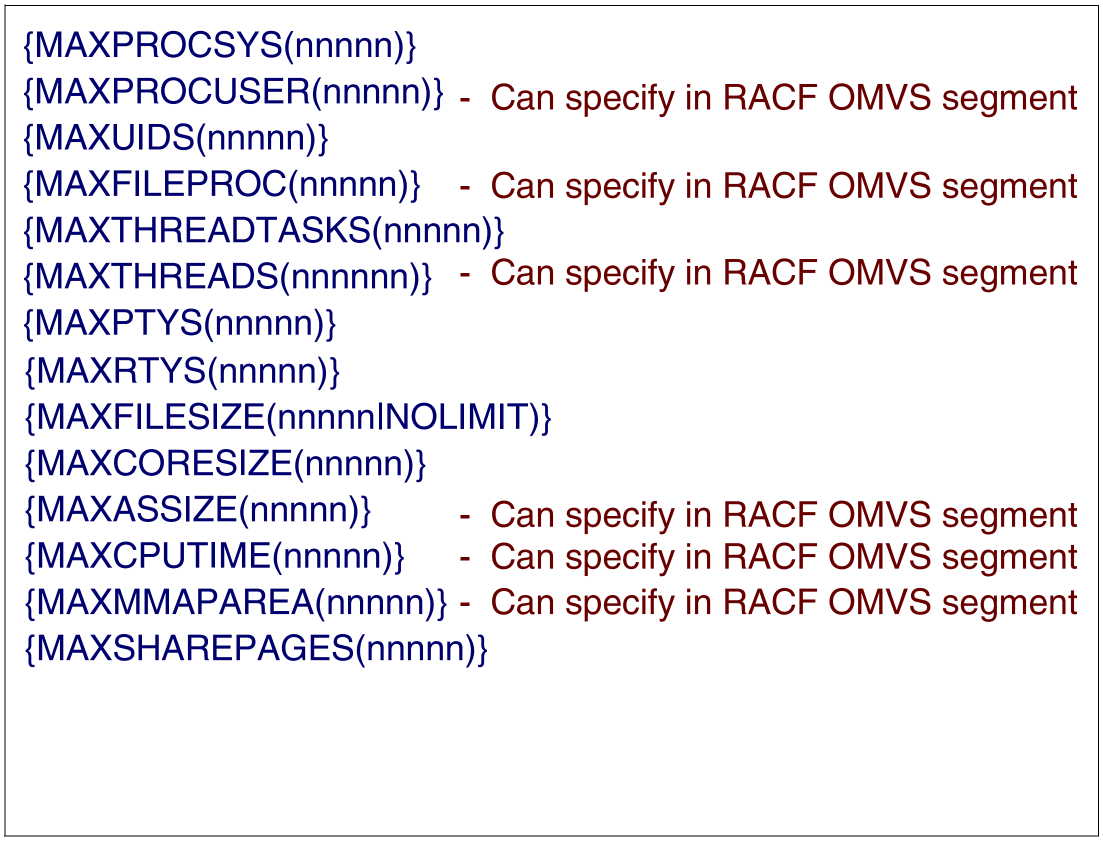

{MAXPROCSYS(nnnnn)}
{MAXPROCUSER(nnnnn)} - Can specify in RACF OMVS segment
{MAXUIDS(nnnnn)}
{MAXFILEPROC(nnnnn)} - Can specify in RACF OMVS segment
{MAXTHREADTASKS(nnnnn)}
{MAXTHREADS(nnnnnn)} - Can specify in RACF OMVS segment
{MAXPTYS(nnnnn)}
{MAXRTYS(nnnnn)}
{MAXFILESIZE(nnnnn|NOLIMIT)}
{MAXCORESIZE(nnnnn)}
{MAXASSIZE(nnnnn)} - Can specify in RACF OMVS segment
{MAXCPUTIME(nnnnn)} - Can specify in RACF OMVS segment
{MAXMMAPAREA(nnnnn)} - Can specify in RACF OMVS segment
{MAXSHAREPAGES(nnnnn)}

Figure 10-4 BPXPRMxx members that control active processes

Figure 10-4 shows all BPXPRMxx Parmlib member parameters you can set up to influence user logon, active processes, file handling, and storage requirements.

You should handle these parameters with care because the values you specify for these statements are interrelated. For example:

- MAXPROCSYS
- MAXPROCUSER
- MAXUIDS

You should always monitor all MAXxxx settings before you change any statements. It makes no sense to allow more users to access UNIX System Services when you do not provide enough TTYs. Each user or process entering UNIX System Services needs a pseudo-terminal (pseudo-TTY).

Beginning in OS/390 V2R8, you can specify the parameters shown in the OMVS segment of a user profile.

For example, if you specify MAXPROCSYS, that means define the maximum number of processes that can be active at the same time, using 3000, and in the MAXPROCUSER statement you allow 200 processes per user and you allow 3000 users to be active at the same time (MAXUIDS), this would not fit.

10.5 Control the number of processes

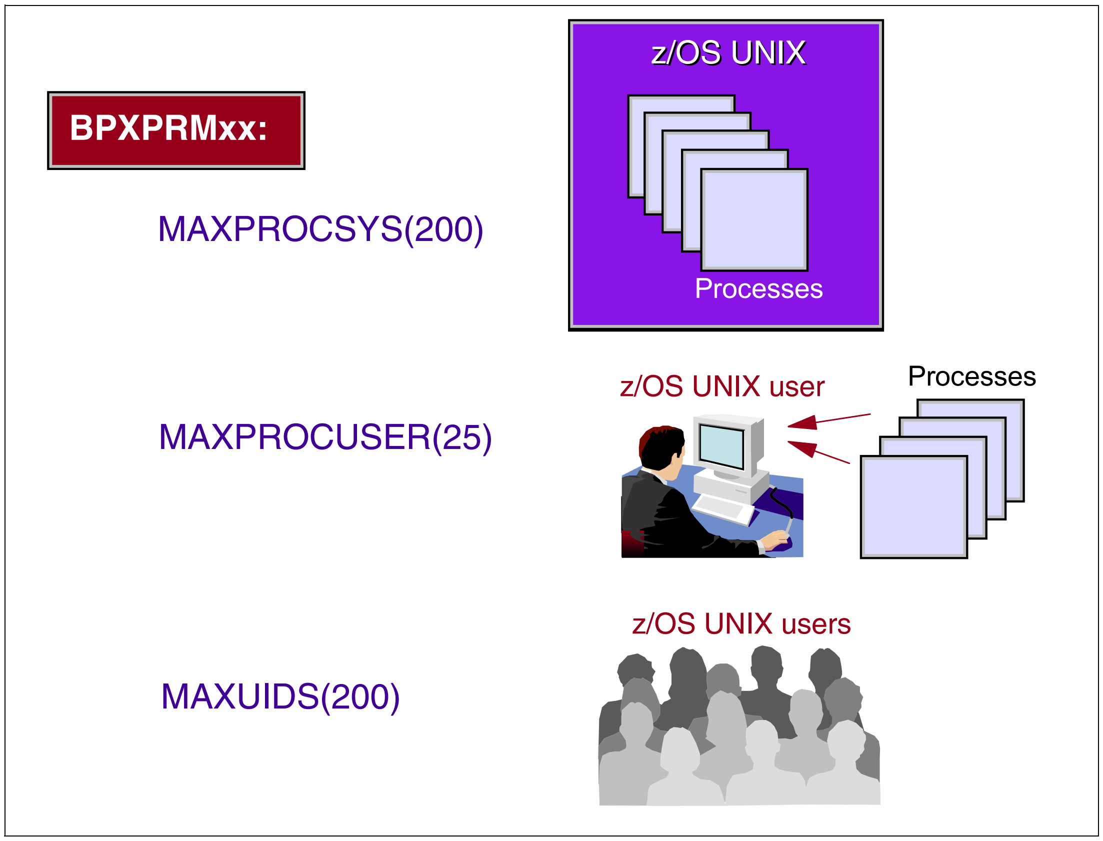


Figure 10-5 BPXPRMxx members that control the number of processes

The **MAXPROCSYS** statement specifies the maximum number of processes that z/OS UNIX will allow to be active at the same time in the system. The range is 5 to 32767; the default is 200. Specify only the maximum number of processes you expect z/OS UNIX MVS to support, because:

- Some storage allocations in the kernel address space are based on the MAXPROCSYS value. A larger value means that more pageable storage will be allocated.
- Most z/OS UNIX MVS processes use an MVS address space, which uses storage. Therefore, avoid specifying a higher value for MAXPROCSYS than the system can support.
- If MAXPROCSYS is set too high relative to the maximum number of initiators the Workload Manager (WLM) can provide, response time delays or failures for fork() or spawn() could occur because users have to wait for initiators.

The **MAXPROCUSER** statement specifies the maximum number of processes that a single OMVS user ID (UID) is allowed to have active at the same time, regardless of how the process became a z/OS UNIX MVS process. The range is 3 to 32767; the default is 25.

If system resources are constrained, setting a low MAXPROCUSER value will limit a z/OS UNIX user's consumption of processing time, virtual storage, and other system resources.

Note: PROCUSERMAX can be specified in the OMVS segment of a user profile to set the maximum number of processes for this UID.

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

The **MAXUIDS** statement specifies the maximum number of unique OMVS user IDs (UIDs) that can use z/OS UNIX MVS services at the same time. The UIDs are for interactive users or for programs that requested z/OS UNIX MVS services. The range is 1 to 32767; the default is 200.

10.6 Resource limits for processes

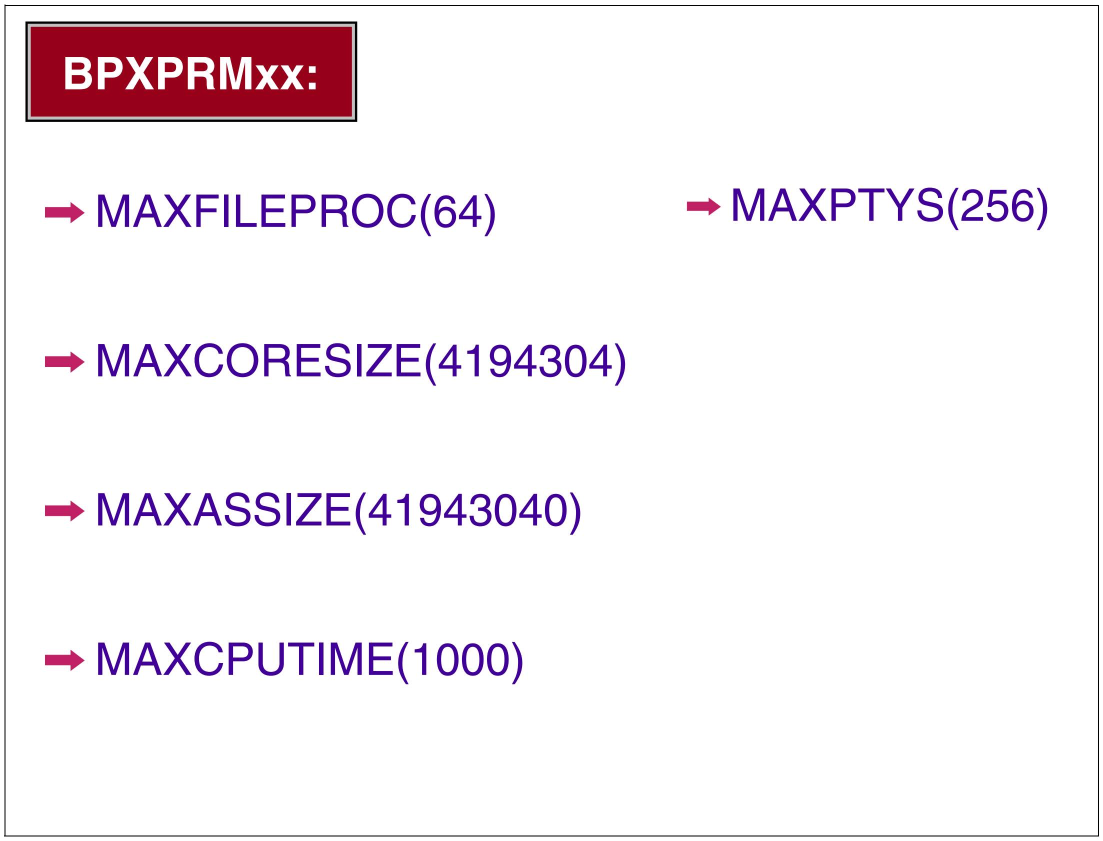


Figure 10-6 Keywords that control resource limits for processes

Some of the system resource limits assigned to processes can be modified by a process using the **`setrlimit`** callable service. Only an authorized process can change the limits beyond what is specified in BPXPRMxx. A resource limit is a pair of values; one specifies the current (soft) limit and the other a maximum (hard) limit. The values assigned in the BPXPRMxx which can be changed by setrlimit are:

- The **MAXFILEPROC** statement specifies the maximum number of files that a single z/OS UNIX user is allowed to have concurrently active or allocated. The range is 3 to 65535; the default and the value in BPXPRMXX is 64.
  - The minimum value of 3 supports the standard files for a process: stdin, stdout, and stderr.
  - The value needs to be larger than 3 to support shell users. If the value is too small, the shell may issue the message `File descriptor not available`. If this message occurs, increase the MAXFILEPROC value.

  Use the RACF **`ADDUSER`** or **`ALTUSER`** command to specify the FILEPROCMAX limit on a per-user basis as follows:

  ```
  ALTUSER userid OMVS(FILEPROCMAX(nnnn))
  ```
- The **MAXCORESIZE** statement specifies the maximum core dump file size (in bytes) that a process can create. The default is 4 MB. If 0 is specified, no core files will be created by the process.
- The **MAXASSIZE** statement indicates the address space region size. The range is from 10 MB to 2 GB. The default is 40 MB. If there are multiple processes within an address

space, the processes share the same limits for MAXASSIZE. Use the RACF **ADDUSER** or **ALTUSER** command to specify the ASSIZEMAX limit on a per-user basis as follows:

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

- The **MAXCPUTIME** is the time limit (in seconds) for processes that were created by rlogind 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-user basis as follows:

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

- 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. But, because each user may have more than one session, it is recommended that you allow four pseudo-TTY pairs for each user (MAXUIDS * 4). Specify a MAXPTYS value that is at least twice the MAXUIDS value.

10.7 Memory mapped files

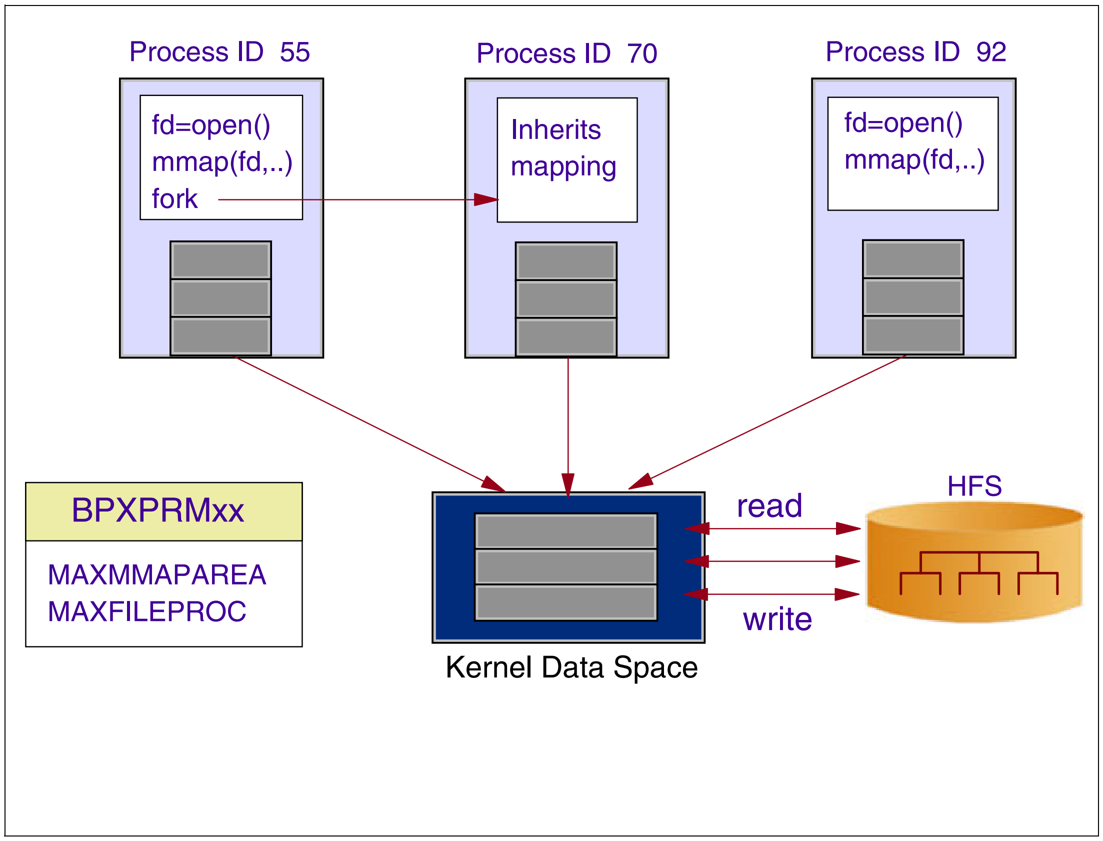


Figure 10-7 BPXPRMxx members for memory mapped files

Memory mapped files allow multiple processes to share data in a file that is mapped in storage. Access to the file is through address space manipulation instead of read/write services. The z/OS UNIX kernel stores the pages of data from the file in a data space. A child process will inherit the mapping addresses from the parent and thereby have access to the data. Other processes will need to open the file and can then map to the same data pages in the data space.

There are two BPXPRMxx parameters which control the use of memory mapped files:

- **MAXFILEPROC** specifies how many files a user can have open or allocated; it also controls the number of memory mapped files a user can have open.
- **MAXMMAPAREA** specifies the maximum amount of data space storage (in pages) that can be allocated for memory mapped files. Use the RACF **ADDUSER** or **ALTUSER** command to specify the MMAPAREAMAX limit on a per user basis as follows:

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

Using memory mapped files causes system memory to be consumed. For each page (KB) that is memory mapped, 96 bytes of ESQA are consumed when a file is not shared with anybody else. When a file is shared, each additional user causes 32 bytes of ESQA to be consumed for each shared page. For example, if 4096 pages are used for memory mapped files, the first user will cause 384 KB (96 * 4000) of ESQA to be consumed. Each additional user will cause 128 KB (32 * 4000) of ESQA to be consumed. ESQA storage is consumed when the mmap() function is invoked rather than when the page is accessed by the application program.

10.8 Control thread resources

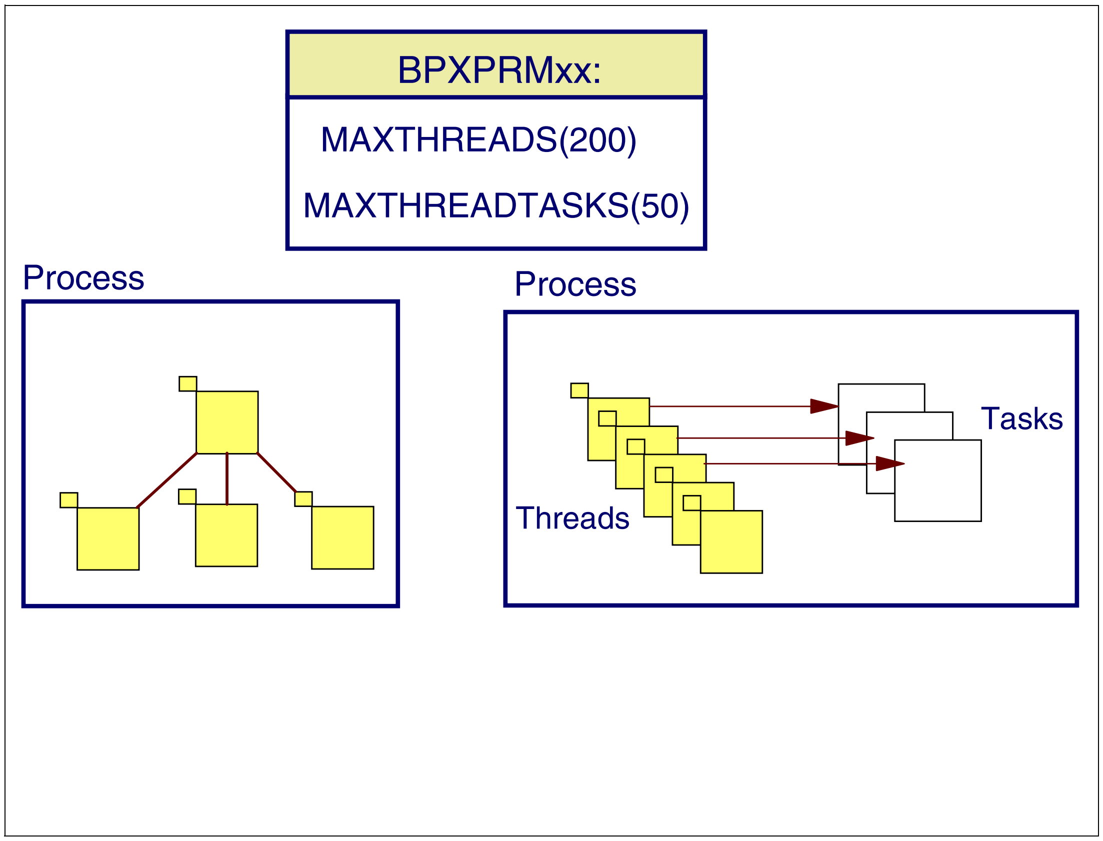


Figure 10-8 Controlling threads

Threads provide support for multiple separate units of dispatchable work within a process. An z/OS UNIX thread can be compared with an MVS task. Threads allow for concurrent and asynchronous processing without the additional overhead associated with creating a new address space.

The thread support is intended for multitasking server applications that require multiple concurrent execution streams. A program using threads can have significant performance benefits in a multiprocessor environment where each thread in a process can run on an individual processor.

- The **MAXTHREADS** statement specifies the maximum number of pthread_created threads, including those running, queued, and exited but not detached, that a single process can have currently active. Specifying a value of 0 prevents applications from using pthread_create. The range is 0 to 100000; the default is 200.
- The **MAXTHREADTASKS** statement specifies the maximum number of MVS tasks created with pthread_create (BPX1PTC) that a single user may have concurrently active in a process. The range is 0 to 32768; the default is 50.

  MAXTHREADTASKS lets you limit the amount of system resources available to a single user process.

Individual processes can alter these limits dynamically if they have sufficient authority.

MVS Tasks and Threads: Each thread that is created with pthread_create runs as an MVS subtask of the initial pthread-creating task (IPT). The IPT is the task that issued the first pthread_create call within the address space. When all the threads created with pthread_create and the IPT have ended, the next task in the address space to issue a pthread_create call is made the IPT. The IPT in the example in Figure 10-8 is the thread running Main.

The difference between these two keywords is that MAXTHREADS specifies the limit for how many threads a process can have active. This number includes the number of threads that are executing on MVS tasks and the number that are waiting for execution. MAXTHREADTASKS specifies the limit for how many MVS tasks can be created per process to schedule threads. This number limits the number of threads that can be executing at the same time in a process. In the pictured example, a process has created 5 threads, but only three MVS tasks, so two threads will be queued until a task becomes available.

10.9 Create a process using fork()

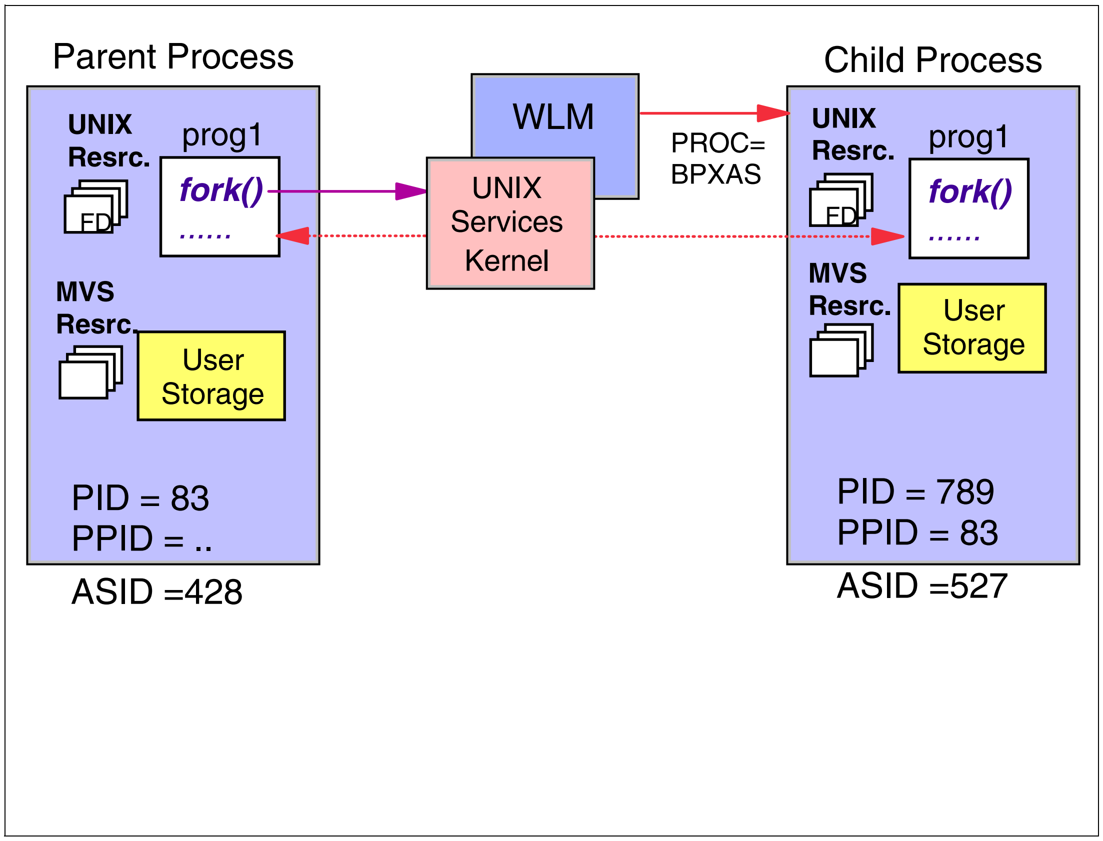


Figure 10-9 Creating a process using the fork() function

Fork() is a POSIX/XPG4 function that creates a duplicate process referred to as a *child* process. The process that issues the fork() is referred to as the *parent* process.

With z/OS UNIX, a program that issues a fork() function creates a new address space which is a copy of the address space where the program is running. The fork() function does a program call to the kernel, which requests WLM to create the child process address space. The storage contents of the parent address space are then copied to the child address space.

After the fork() function completes, the program in the child AS will start at the same instruction as the program in the parent AS. Control is returned to both programs at the same point. The only difference that the program sees is the return code from the fork() function. A return code of zero is returned to the child after a successful fork(). The return code for the parent is the child process ID (PID).

Also, any UNIX resources (pipes, sockets, files) accessible via opened File Descriptors in the parent are propagated to the new address space. z/OS resources such as DD allocations, cross-memory resources, and ENQ serializations are *not* propagated to the child address space.

The program prog1 in the parent AS issues the fork() function, the kernel is called to perform the action. The kernel calls the Workload Manager (WLM) to create a new address space. WLM uses the BPXAS procedure from PROCLIB to initialize the address space. WLM dynamically manages the number of address spaces started for UNIX processes, in order to meet goals set via ICS/IPS, or WLM policy in goal mode.

Once the child address space has been created, the child gets the required storage from a STORAGE request. The kernel then copies the contents of the parent AS to the child AS using the MVCL instruction. After some additional setup, the kernel returns codes to both parent and child programs. The program in the child AS gets control at the same point as the program in the parent AS. The only difference is the return code from the fork() function.

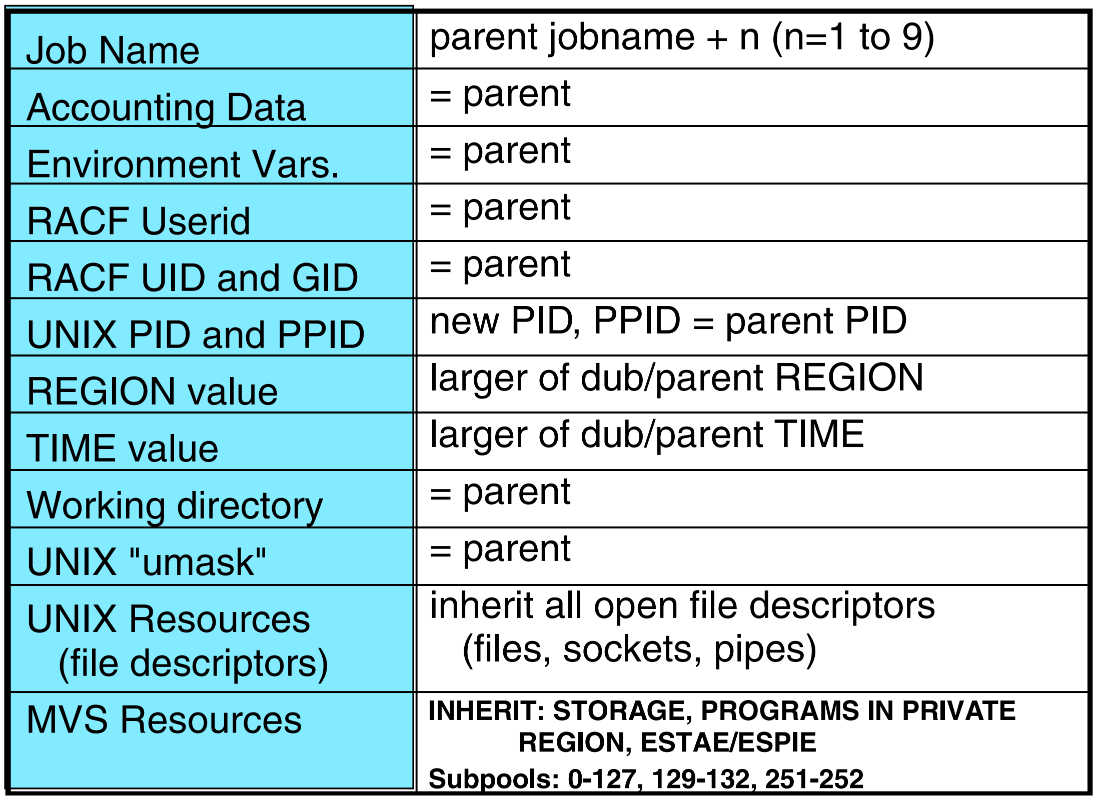
The child address space is almost an identical copy of the REGION storage in parent address space. User data, for example private subpools, and system data, like RTM (Recovery Termination Management) control blocks, are identical.

HFS files are allocated to the kernel and I/O is done by calling the kernel. An open HFS object (file, pipe, socket) for the parent is represented by an open File Descriptor in the File Descriptor Table. On a fork(), all open HFS File Descriptors (that is, files) are inherited by the child address space. The child does not inherit any file locks for the HFS files.

z/OS resources are not propagated to the child address space. Any linkage stack in the parent is not carried over to the child. Also, data spaces and hiperspaces are not carried over since access list and access register contents are not copied. Internal timers together with SMF and SRM accounting data are set to zero in the child address space.

Also, z/OS data sets are allocated to an individual address space, and after a fork() the child address space does not have access to the z/OS data sets allocated by the parent. Also, ENQ resources held by the parent are not propagated.

10.10 Values for forked child process



Job Name	parent jobname + n (n=1 to 9)
Accounting Data	= parent
Environment Vars.	= parent
RACF Userid	= parent
RACF UID and GID	= parent
UNIX PID and PPID	new PID, PPID = parent PID
REGION value	larger of dub/parent REGION
TIME value	larger of dub/parent TIME
Working directory	= parent
UNIX "umask"	= parent
UNIX Resources (file descriptors)	inherit all open file descriptors (files, sockets, pipes)
MVS Resources	**INHERIT: STORAGE, PROGRAMS IN PRIVATE REGION, ESTAE/ESPIE** **Subpools: 0-127, 129-132, 251-252**

Figure 10-10 Propagation of parameters passed to the child process

JOBNAME of the parent is propagated to the child and appended with a numeric value in the range of 1-9 if the jobname is 7 characters or fewer. If the jobname is 8 characters, it is propagated as is. Numeric count wraps back to 1 when it exceeds 9.

z/OS accounting data is copied from parent to child. UNIX environment variables are copied from parent to child. RACF security profile (Userid, UID, GID) is also inherited from the parent.

The child is allocated a new process ID (PID). Child PPID = parent PID.

The child REGION size is set by the “dub” process. The current REGION size of the parent is compared with BPXPRMxx MAXASSIZE, and the larger value is set as REGION size.

Child TIME value is also set by the “dub” process. The current TIME for the parent task is compared with BPXPRMxx MAXCPUTIME, and the larger value is set as the TIME value.

Current working directory and umask value for the parent are propagated to the child.

All resources defined by open file descriptors are propagated to open file descriptors on the child.

The only z/OS resources that are propagated to the child are the subpools in the parent REGION area (this includes loaded programs and data), plus the recovery environment for loaded programs established via ESTAE/ESPIE calls.

10.11 Starting a program with exec()

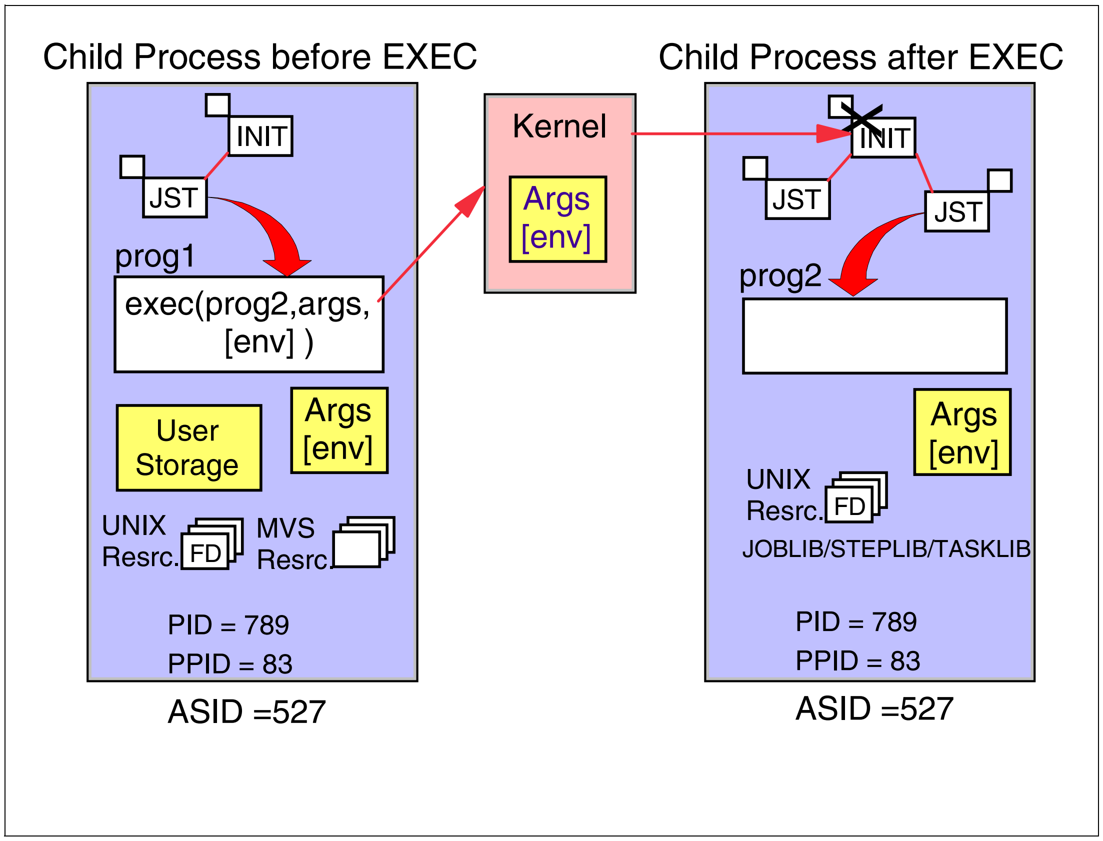


Figure 10-11 Creating a process using the exec() function

The z/OS UNIX environment provides a family of **exec()** function calls. These calls load a new program and transfer execution from the calling program to the new program. The difference between the exec calls is based on how the pathname of the new program is specified, and whether the value of environment variables will be passed.

When a program issues an exec() call, it passes these parameters:

- Program name - either partial/full pathname or a file name only
- A series of “arguments” (parameters) to be passed to the new program
- Optionally, an array of environment variables are passed via parm env()

When the kernel receives an exec call, it erases all the current storage contents of the address space. It also terminates the current process task, and starts a new one. The new program is loaded from the HFS, and the values of the arguments and optional environment variables are placed in storage. The new program starts up using only this basic information.

No z/OS resources are propagated from the caller's TCB to the new TCB except JOBLIB, STEPLIB, and TASKLIB. However, open file descriptors from the exec() caller are propagated to the new program, giving it access to resources established by the caller.

An exception to no z/OS resources being propagated is that the caller's task JOBLIB/STEPLIB/TASKLIB status may be propagated to the new task. This requires the caller to set the STEPLIB variable before exec(). (We explain more about this later.)

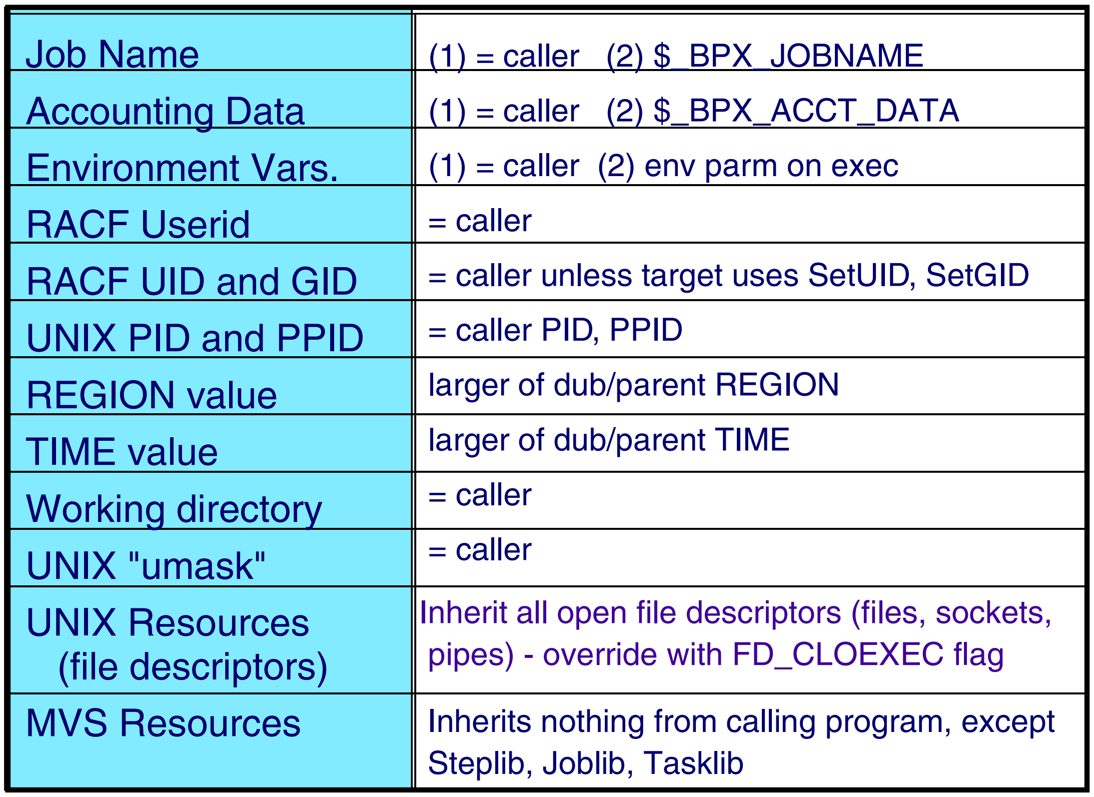
If exec() is used by a process running in an address space created by fork or spawn, the caller of exec() *must* be running under the job step task TCB, with no subtasks, and with no linkage stack. Any other situation will cause an ABEND EC6. For processes created in multi-task address spaces via attach_exec, execmvs, or attach_execmvs, this restriction is relaxed.

The fork() and exec() calls are usually used together to create a new address space running a child process, and then pass control to a new program in the child. There are some exceptions to this:

- A C program started from a z/OS batch job could use the exec() call to start a program from the HFS. The new program would then replace the original C program executed in batch.
- The BPXBATCH utility can be used to run shell scripts, or C programs resident in the HFS, as a batch job in a JES initiator. After the BPXBATCH job step program has set up the support environment, it then issues exec() to replace BPXBATCH with the program or shell script to be run.

Note: The target of an exec() can be a shell script or REXX EXEC. In this case, the new program loaded by the kernel is the SHELL program, and the script or REXX EXEC is automatically executed under the shell.

10.12 Values passed for exec() program



Job Name	(1) = caller (2) $_BPX_JOBNAME
Accounting Data	(1) = caller (2) $_BPX_ACCT_DATA
Environment Vars.	(1) = caller (2) env parm on exec
RACF Userid	= caller
RACF UID and GID	= caller unless target uses SetUID, SetGID
UNIX PID and PPID	= caller PID, PPID
REGION value	larger of dub/parent REGION
TIME value	larger of dub/parent TIME
Working directory	= caller
UNIX "umask"	= caller
UNIX Resources (file descriptors)	Inherit all open file descriptors (files, sockets, pipes) - override with FD_CLOEXEC flag
MVS Resources	Inherits nothing from calling program, except Steplib, Joblib, Tasklib

Figure 10-12 Parameters passed to the new process

Due to the creation of a new process task for the exec() program, the scheduling environment can change considerably over an exec() call, as follows:

- By default, the address space JOBNAME stays the same. However, an authorized caller (a superuser running in a forked address space) can set a new name envar $_BPX_JOBNAME before exec() - this will change the address space JOBNAME.
- By default, account data is inherited. Alternatively, any exec() caller an set envar $_BPX_ACCT_DATA to new account data prior to issuing a call.
- By default, a new program inherits all caller environment variables. You can use the env() parameter to pass *only* chosen values to the new program.

The RACF Userid remains constant over exec(). UID and GID are also constant, unless the target program has SetUID or SetGID flags set. In this case, the effective UID or GID is changed for the new program. The PID and PPID values are not changed.

As with fork, REGION and TIME values are first set by “dub” values in BPXPRMxx. The inherited values related to the exec() caller TASK are then compared to the dub values, and if inherited values are larger, they override dub.

Working directory and umask are inherited from the caller.

All open file descriptors are passed across to the new program for immediate use, except those previously flagged with an FD_CLOEXEC flag.

No z/OS resources are inherited, except possibly JOBLIB/STEPLIB.

10.13 z/OS UNIX processes get STEPLIBs

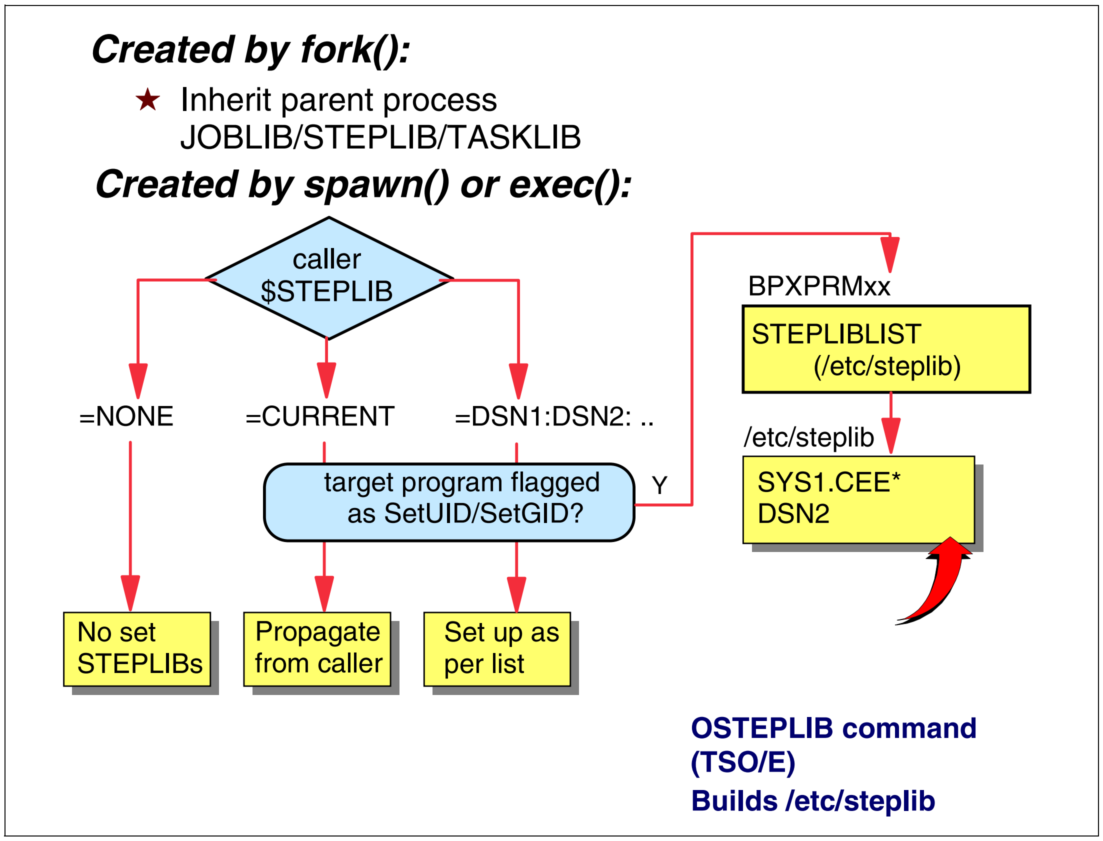


Figure 10-13 The use of STEPLIBs when creating a new process

Figure 10-13 illustrates that programs requested by a UNIX application can be loaded from z/OS program libraries, including STEPLIBs.

For a process that has been created by a fork() request, the child process inherits any active parent process JOBLIB/STEPLIB concatenation.

When a process wants to use a `spawn()` or an `exec()` call, the caller must set the $STEPLIB envar prior to executing the fork()/spawn() request so that the kernel can control the STEPLIB allocation. The settings have the following meanings:

- **CURRENT** - Propagate and reuse JOBLIB/STEPLIB/TASKLIB from task TCB issuing the exec/spawn to the new process/program.
- **NONE** - No STEPLIB; do not propagate from the caller if present.
- **DSN1:.DSN2: ..** - Set this as STEPLIB concatenation.

If the $STEPLIB variable is not set, the default assumed is CURRENT.

If the target program for the spawn() or exec() function has the SetUID or SetGID flag set, a further check is made against the sanctioned library list. This list is pointed to by the BPXPRMxx statement **STEPLIBLIST**. Libraries in the JOBLIB/STEPLIB list to be propagated must also be named in the sanction list. If a library is not sanctioned, it will be dropped from the propagated STEPLIB list.

10.14 Locating programs for z/OS UNIX processes

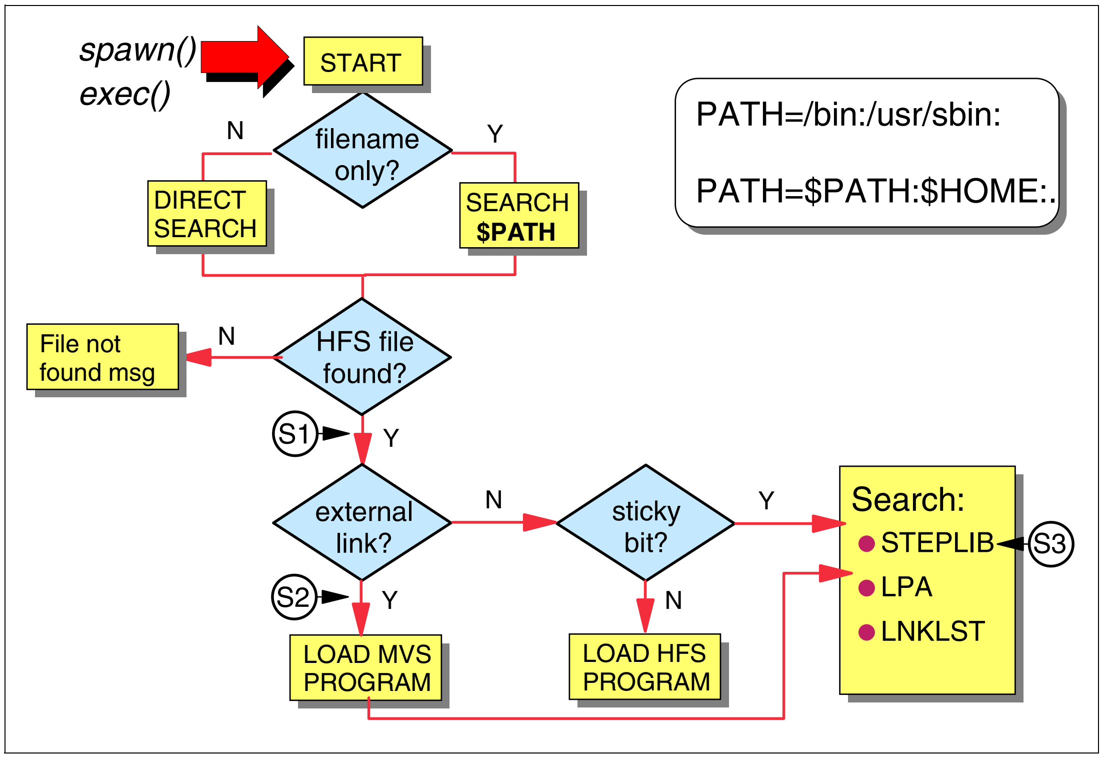


Figure 10-14 Loading programs for z/OS UNIX processes

The `spawn()` and `exec()` calls both request z/OS UNIX to find and load a new HFS program. Figure 10-14 shows the overall flow. First scan the supplied pathname of the HFS program for imbedded "/" characters, with the following results:

- First char = "/" - Supplied name is the absolute pathname; search HFS directly.
- Contains one or more "/," not in position 1. Treat this as a relative pathname by suffixing with current working directory, then search HFS for full name.
- No "/" - Supplied name is filename; search directory concatenation supplied in $PATH environment variable, use first object with a matching name.

S1 If the HFS file is located, check permission bits for user execute auth.

If access is granted, check to see if object is an external link. If so, attempt to load the z/OS program named in the link.

S2 is a second security access check. It is made based on RACF user ID and the DSNAME.

If the object is not an external link, check to see if the sticky bit is on. If so, switch to do a z/OS search for the program. Otherwise attempt to load and execute the HFS object as a program. For a z/OS search, follow the standard search sequence.

S3 Search any JOBLIB/STEPLIB concatenation pointed to by the process task.

If an object is found, make a security check based on RACF user ID and DSNAME before loading the program.

- Search the LPA areas.
- Search the system LNKLST.

10.15 Shared pages for the fork() function

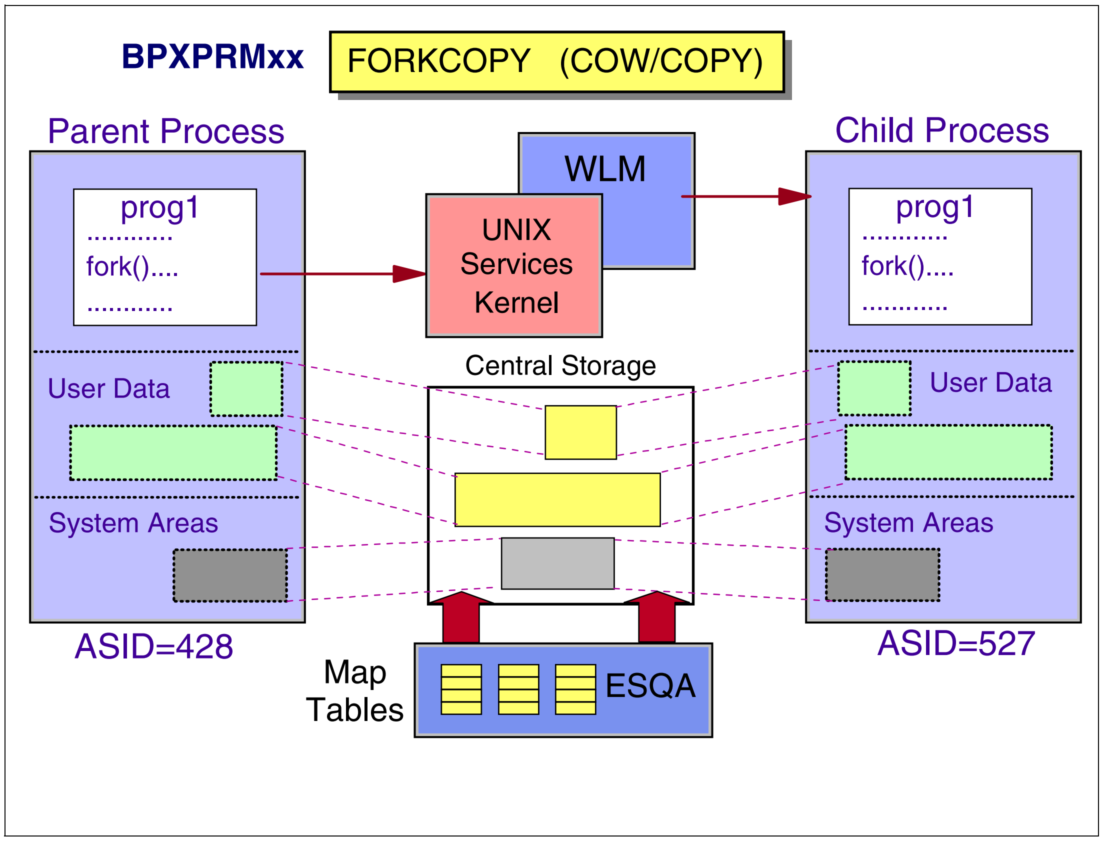


Figure 10-15 Shared pages for the fork() function

The shared pages function provides the capability to define virtual storage areas through which data can be shared by programs within or between address spaces or data spaces. Sharing reduces the amount of processor storage required and the I/O necessary to support data applications that require access to the same data.

The z/OS UNIX fork function can use shared pages to improve performance. The keyword `FORKCOPY(COW|COPY)` in the BPXPRMxx member specifies how storage is copied from the parent process to the child during fork.

FORKCOPY(COW)	Copy On Write (COW). Storage areas will be shared between the parent and child, and they will only be copied to the child if they are modified by either the parent or the child. This requires the suppression-on-protection (SOP) hardware feature.
FORKCOPY(COPY)	All storage areas will be copied from the parent to the child. Shared pages will not be used.

FORKCOPY(COW) is the default. Figure 10-15 illustrates how the storage area of a parent process is copied to a child process during a fork operation. For a UNIX system this is not a big deal, but for an MVS system it is a very costly operation. In most cases where a fork function is used, it is followed by an exec function in the child. This means that a new program is started in the child and all the storage areas that were copied from the parent are released.

10.16 Spawn function

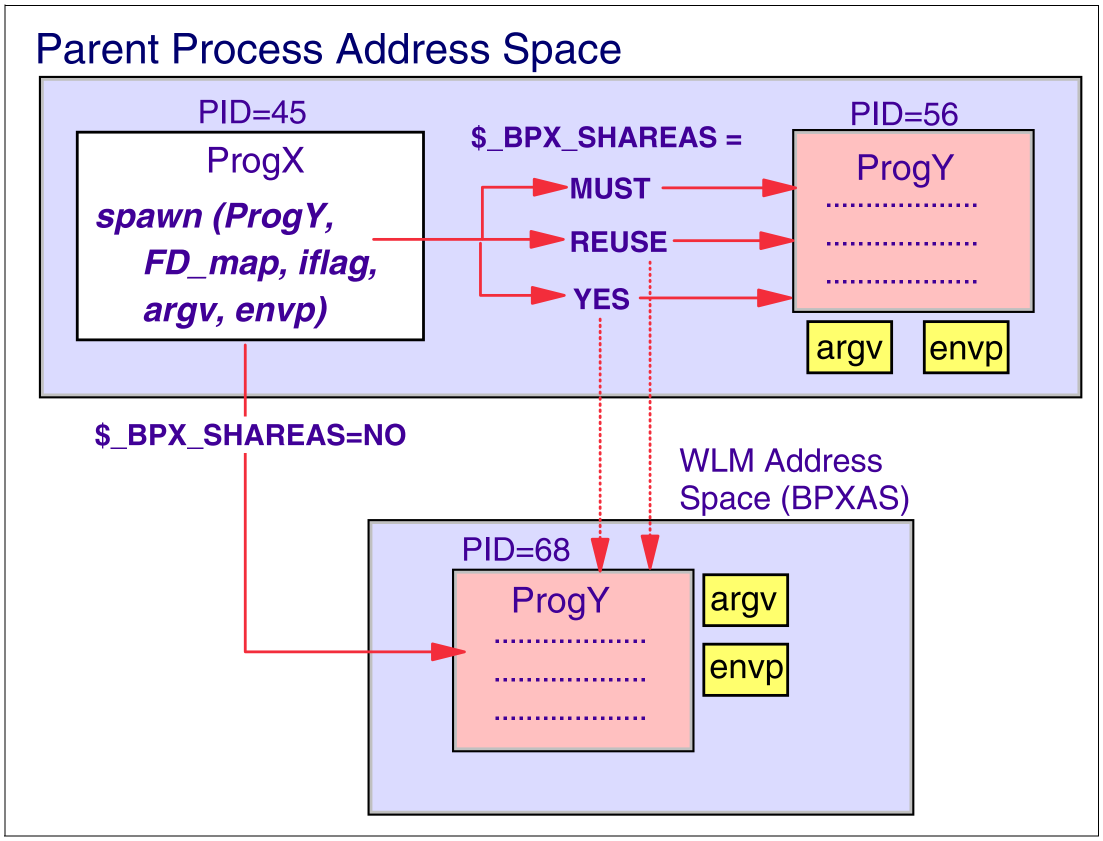

Parent Process Address Space
PID=45
ProgX
spawn (ProgY, FD_map, iflag, argv, envp)
$_BPX_SHAREAS =
MUST
REUSE
YES
PID=56
ProgY
argv
envp
$_BPX_SHAREAS=NO
WLM Address Space (BPXAS)
PID=68
ProgY
argv
envp

Figure 10-16 The spawn function

Spawn is a combination of fork and exec since spawn will create a new process and start a new program in the child process.

Spawn can create a child process in the same address space as the parent process or in a new address space. An environment variable called _BPX_SHAREAS will decide where the child process will be created.

- **_BPX_SHAREAS=YES** - The child process will be created in the same address space as the parent. The benefits of using BPX_SHAREAS=YES are:
  - The spawn runs faster
  - The child process consumes fewer system resources.
  - The system can support more resources.

  If YES fails, a new address space is created.

  The side effects are:
  - When running multiple processes with BPX_SHAREAS=YES, the processes cannot change identity information. For example, setuid and setgid will fail.
  - You cannot run a setuid or setgid program in the same address space for another process.
  - When the parent terminates, the child will terminate because it is a subtask.

- **_BPX_SHAREAS=MUST** - MUST specifies that the child process must be created on a subtask in the parent's address space. If the request cannot be honored, the request will complete unsuccessfully.
- **_BPX_SHAREAS=REUSE** - REUSE specifies that the child process is to be created on a subtask in the parent's address space and when the process terminates, system structures for the child process are left in place and reused when the parent spawns another process with _BPX_SHAREAS=REUSE.

  Using BPX_SHAREAS_REUSE is similar to specifying YES. In environments where shell commands are invoked over and over, the REUSE option will perform better.
- **_BPX_SHAREAS=NO** - The child process will be created in a separate address space.

_BPX_SHAREAS=NO is the default setting for this environment variable.

10.17 Interprocess communication functions

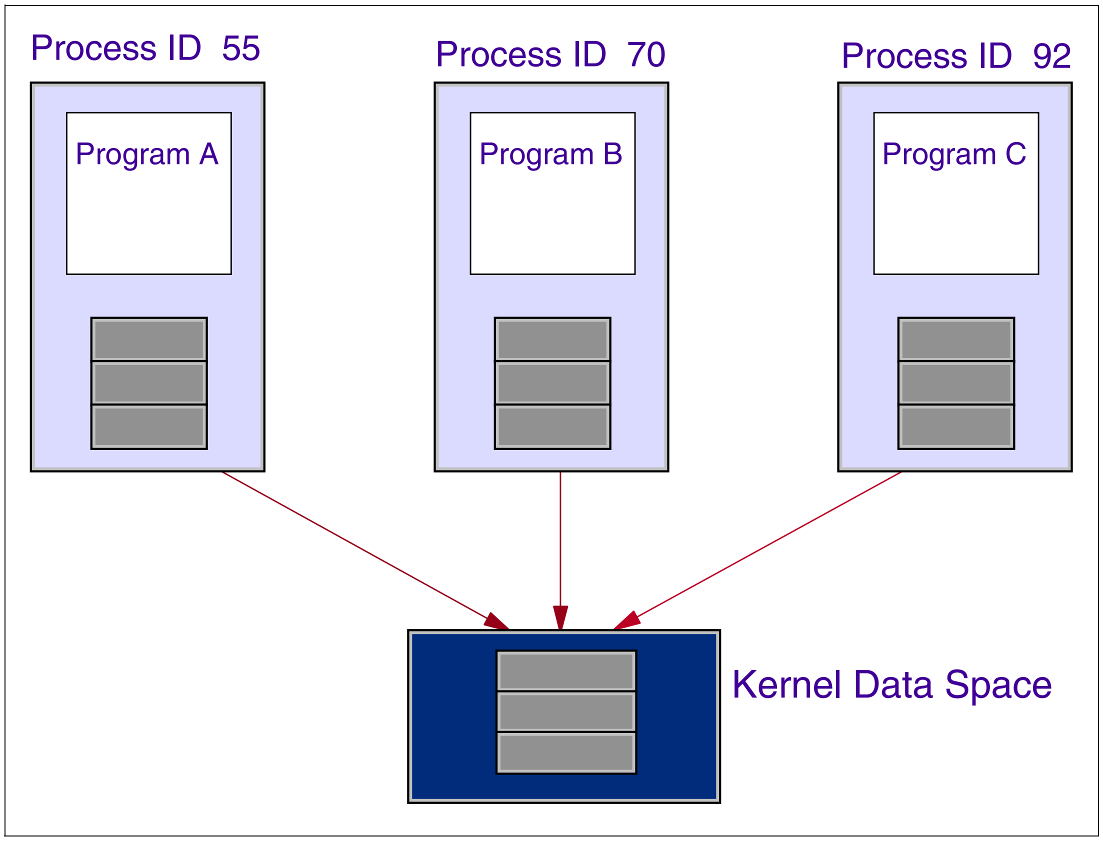


Figure 10-17 Interprocess communication functions

Support for XPG4 introduced Interprocess Communication (IPC) functions in z/OS UNIX. These IPC functions were required by many applications, particularly client/server applications.

Message Queues - Message queues allow a client and a server process to communicate through one or more message queues in the kernel. A process can create, read from, or write to a message queue. Multiple client and server processes can share the same queue.

Shared Memory - Shared memory provides a method of sharing data in storage between multiple processes. The shared data is kept in a data space created by the kernel. The data can be shared between a parent and child process or between unrelated processes.

Semaphores - Semaphores are used for serializing access to shared memory. A program using shared memory must get a semaphore before it allocates shared memory.

10.18 Control IPC resources

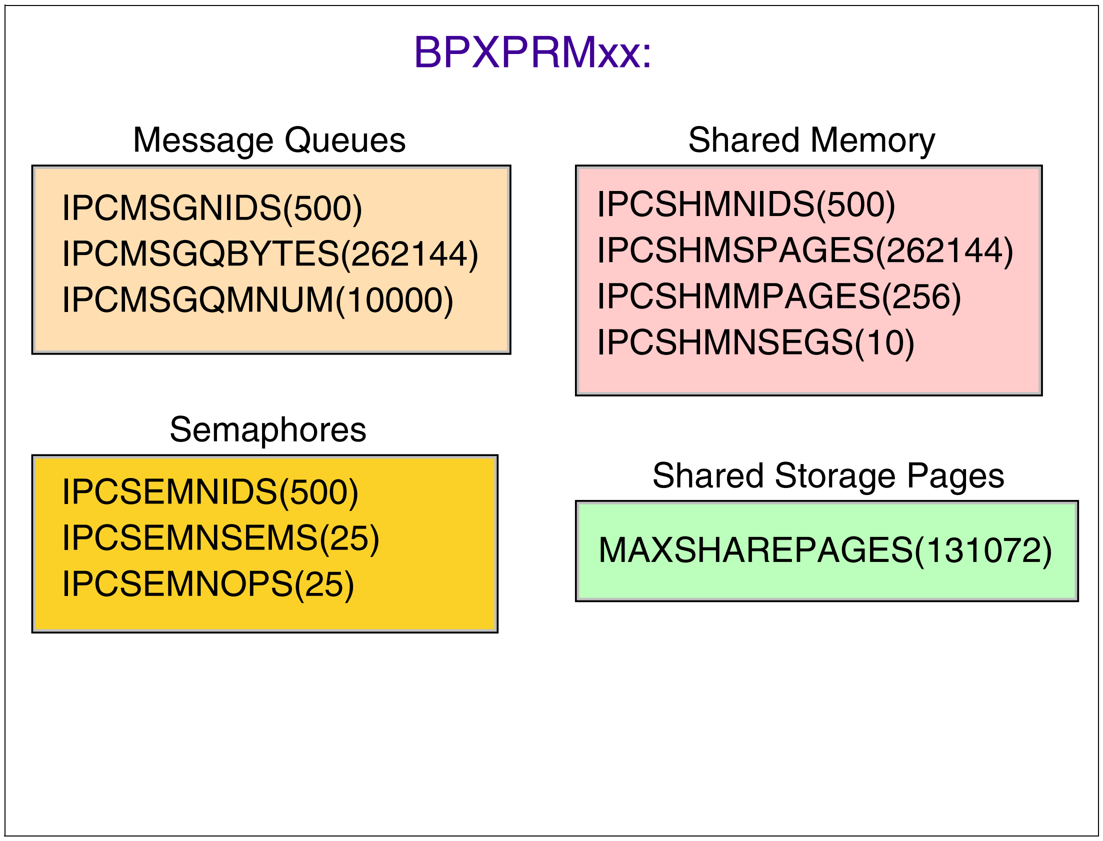


Figure 10-18 Controlling interprocess communication functions

Message queues control:

- **IPCMSGNIDS** - Specifies the maximum number of unique message queues in the system. The range is from 1 to 20000. The default is 500.
- **IPCMSGQBYTES** - Specifies the maximum number of bytes in a single message queue. The range is from 0 to 1048576. The default is 262144.
- **IPCMSGQMNUM** - Specifies the maximum number of messages for each message queue. The range is from 0 to 20000. The default is 10000.

Shared memory control:

- **IPCSHMNID** - Specifies the maximum number of unique shared memory segments in the system. The range is from 1 to 20000. The default is 500.
- **IPCSHMSPAGES** - Specifies the maximum number of pages for shared memory segments in the system. The range is from 0 to 2621440. The default is 262144.
- **IPCSHMMPAGES** - Specifies the maximum number of pages for a shared memory segment. The range is from 0 to 25600. The default is 256.
- **IPCSHMNSEGS** - Specifies the maximum number of shared memory segments attached for each address space. The range is from 0 to 1000. The default is 10.

Semaphore control:

- **IPCSEMNIDS** - Specifies the maximum number of unique semaphore sets in the system. The range is from 1 to 20000. The default is 500.

- **IPCSEMNSEMS** - Specifies the maximum number of semaphores for each semaphore set. The range is from 0 to 32767. The default is 25.
- **IPCSEMNOPS** - Specifies the maximum number of operations for each semaphore operation call. The range is from 0 to 32767. The default is 25. This is a system-wide limit.

Control shared pages:

- **MAXSHAREPAGES**. - Specifies the maximum number of shared storage pages that can be concurrently in use by z/OS UNIX functions. This can be used to control the amount of ESQA consumed, since shared storage pages cause the consumption of ESQA storage. The functions that this limit applies to are fork(), mmap(), shmat(), and ptrace(). The range is from 0 to 32768000. The default is 131072 pages.

10.19 Interprocess communication signals

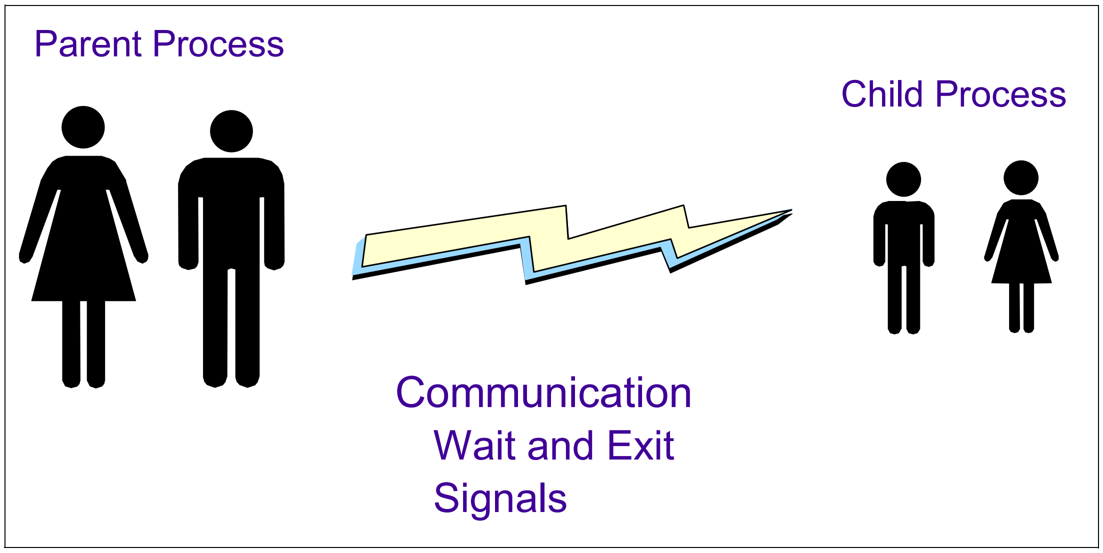


Figure 10-19 Interprocess communication signals

- **Wait and Exit** - A parent process can wait on the exit of a child. The wait() function is used for waiting for any child process, while the waitpid() function is used for waiting for a particular child process.

  Both exit() and _exit() terminate a process and generate status information which is available for the parent process waiting with wait() or waitpid(). When using exit(), these cleanup routines are not invoked. Generally, exit() is used for a graceful exit from a program, while _exit() is used for abnormal terminations.

- **Signal() and sigaction()** are equivalent functions which are used to catch a signal and determine what to do with it.

  The function for sending signals is called kill(). A process can send a signal to another process or group of processes if it has permission to do so. A process can also send a signal to itself.

  Signals are used for system event notification, or they can be used for process synchronization. For example, a process might want to wait for a signal to know that another process has opened a pipe, written a file, or completed a task that the current process needs to wait for.

A process can choose what to do when it receives a signal:

- Execute a signal handling function.
- Ignore the signal.
- Restore the default action of a signal.

Kill() accepts several different signal codes. Examples are:

- SIGABND - abend
- SIGCHLD - child termination
- SIGKILL - cancel a process
- SIGSTOP - stop a process

From a program's point of view, signals are asynchronous. That means a program can, in principle, receive a signal between any two instructions.

10.20 Pipes

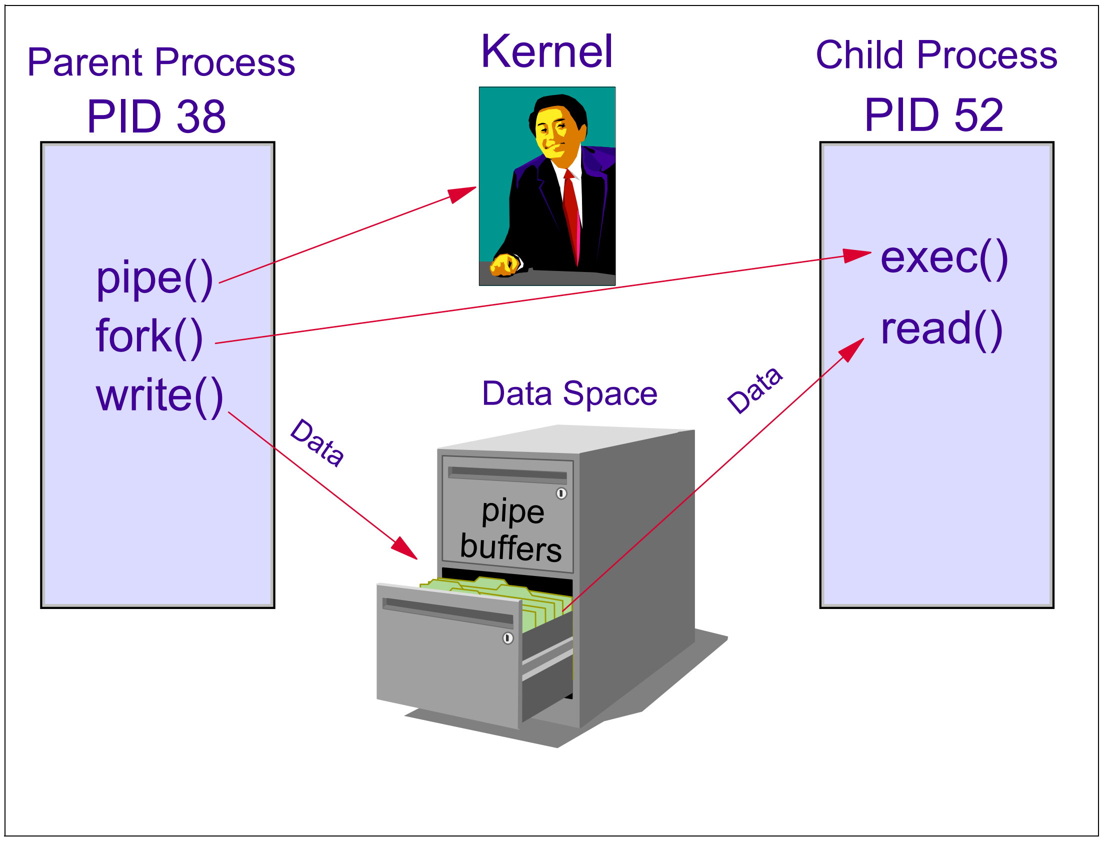


Figure 10-20 Using pipes with z/OS UNIX

In z/OS UNIX, pipes are a communication mechanism for sending arrays of data between two processes. Pipes are conceptually like a sequential file. Processes can write into a pipe and other processes can read from the pipe.

Pipes exist only during the time they are open. The storage needed for the pipes is obtained from a data space associated with the kernel address space.

Data can only be read from the beginning of the pipe. You cannot seek in a pipe. Data is always read from a pipe in the same order it was written into the pipe. The data is discarded when all the processes that read from the pipe have closed the pipe. There are two types of pipes:

- **Unnamed pipes** are used between related processes, for example between a parent and child process. An unnamed pipe is created by the pipe() function. The pipe function also opens the pipe for use. No security checking is performed on unnamed pipes since they are available only to a process and its children.
- **Named pipes** are also called FIFO. A FIFO resides in the hierarchical file system and can be referred to by a name; thus it can be used for communication between any two processes.

10.21 Other BPXPRMxx keywords

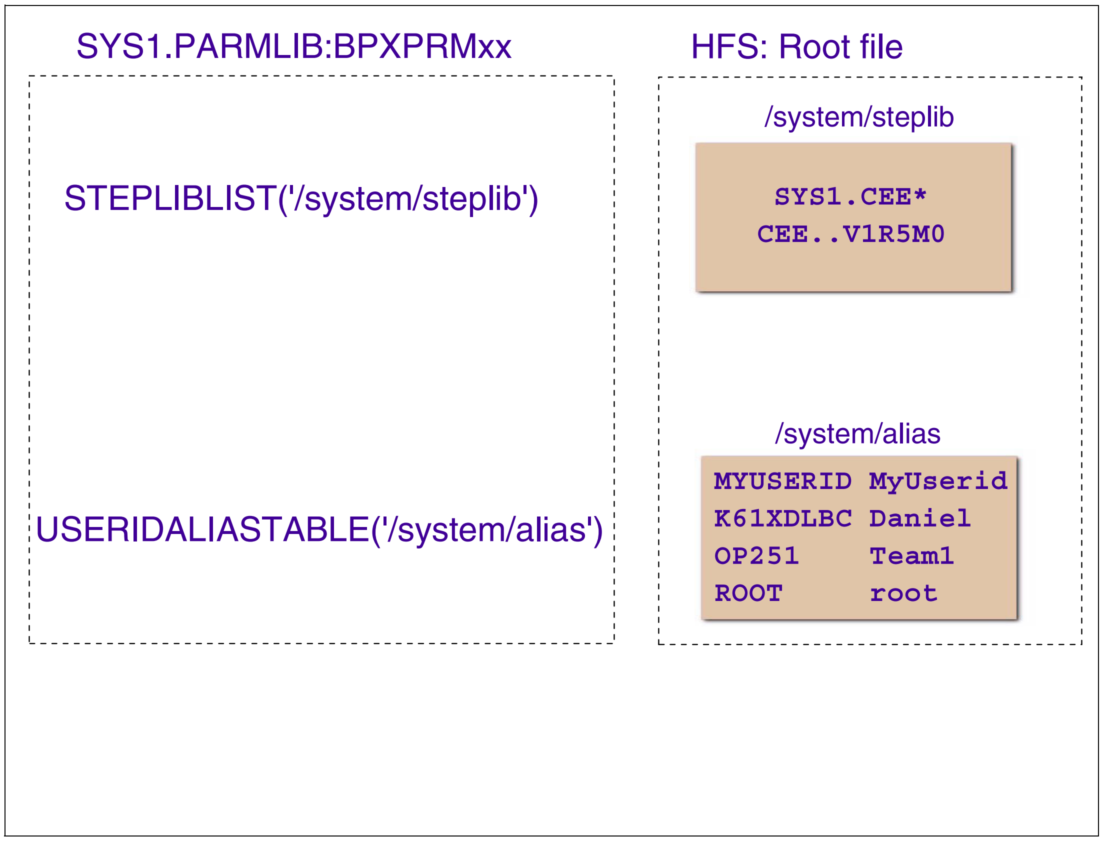


Figure 10-21 Other BPXPRMxx keywords

The **STEPLIBLST** parameter specifies a pathname of a file in the hierarchical file system. This file is intended to contain a list of MVS data sets that are sanctioned by an installation for use as step libraries for programs that have the set-user-ID and set-group-ID permission bits set.

The TSO/E command **`OSTEPLIB`** can be used to build or modify the file which contains the list of MVS data sets that can be step libraries. Superuser authority is required to run this command.

The **USERIDALIASTABLE** statement allows an installation to associate an alias name with an MVS user ID. If specified, this alias name is used in z/OS UNIX processing for the user IDs listed in the table. The USERIDALIASTABLE statement specifies the pathname of a hierarchical file system (HFS) file. This file is intended to contain a list of MVS user IDs and their associated alias names.

Specifying the USERIDALIASTABLE statement causes poorer performance and increases systems management costs and complexity. Installations are encouraged to continue using uppercase-only user IDs.

10.22 More BPXPRMxx keywords

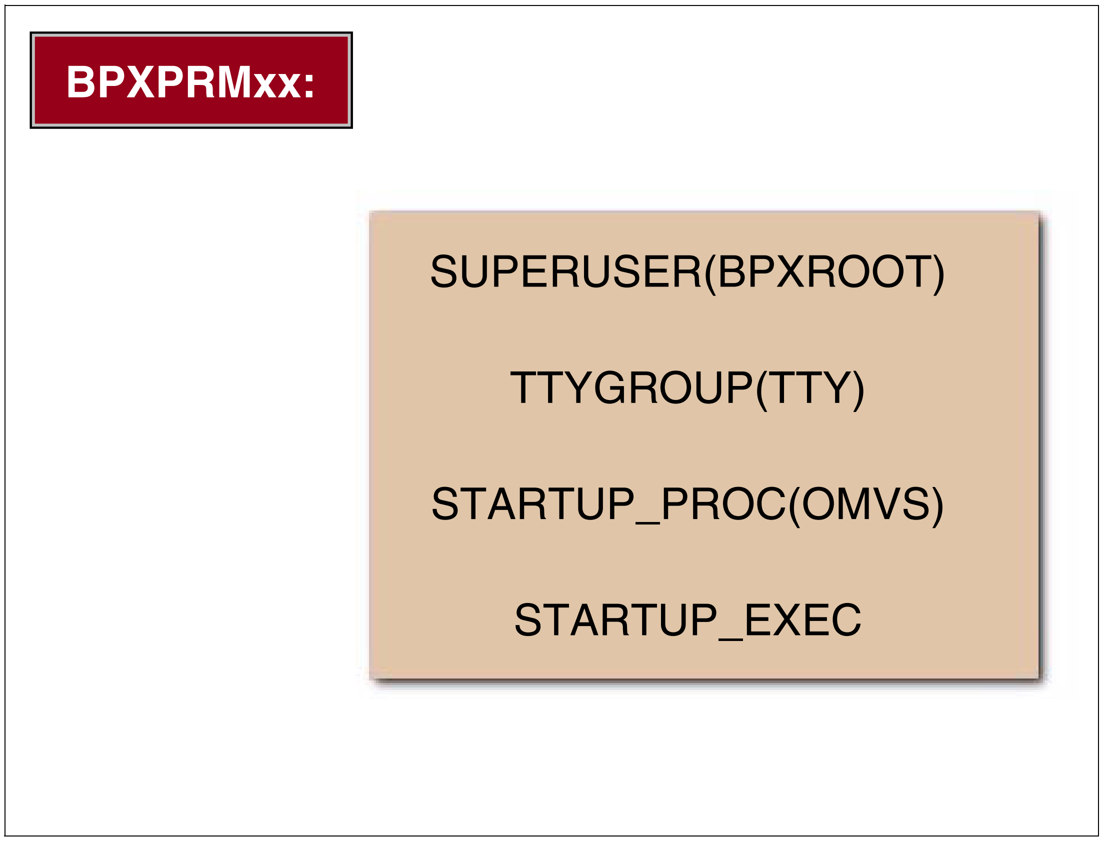


Figure 10-22 More BPXPRMxx keywords

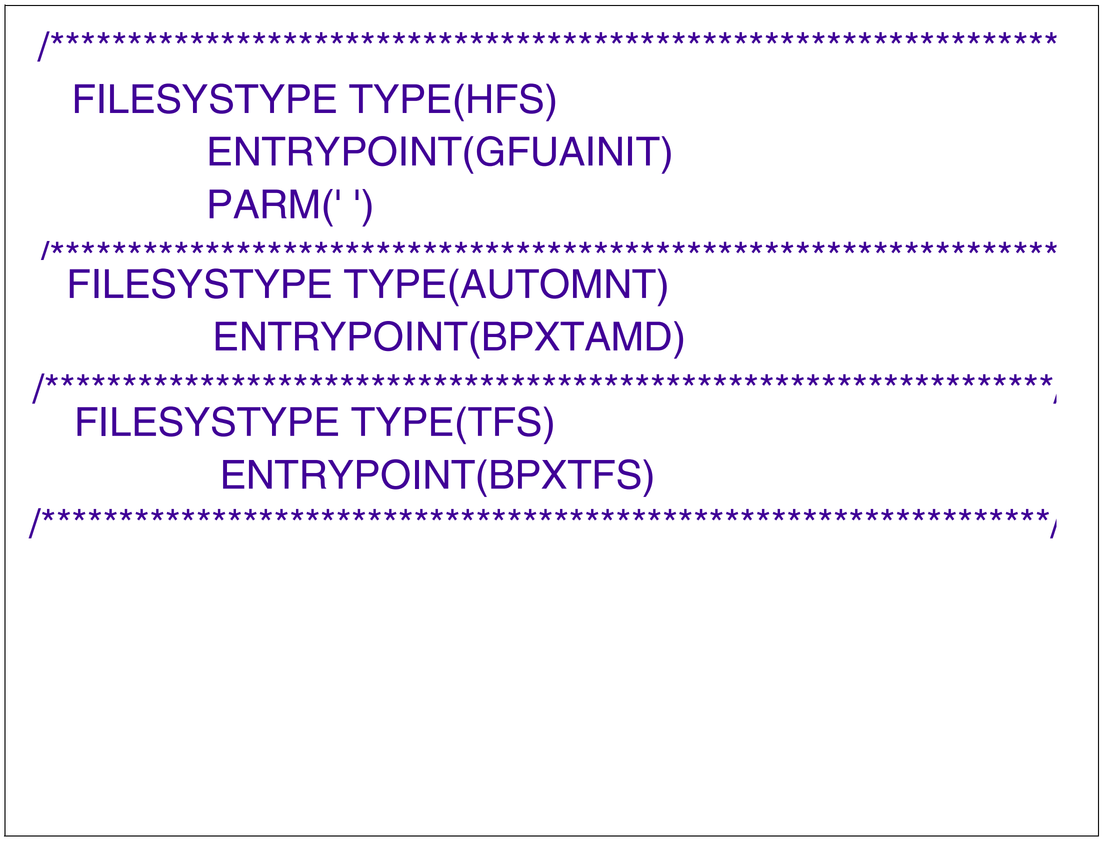
The **SUPERUSER** parameter specifies a superuser name. This will be used when a daemon task issues a setuid() to set a UID of 0 and the user name is not known. It should be an ID defined to the security product with no access to MVS resources and a UID of 0. The default is SUPERUSER(BPXROOT).

TTYGROUP is a group name for the pseudo-terminal files (ptys and rtys) when they are first opened. This group name should be defined to the security product with a GID, but with no users in the group. TTY is the default group name.

The **STARTUP_PROC(OMVS)** is the started JCL procedure that initializes the kernel. If you change this it must be a single step procedure that invokes the BPXINIT program.

The **STARTUP_EXEC** is a startup exec that replaces the /etc/init program that BPXOINIT normally invokes. This is used by installations that want to run with a minimal configuration, but would like to populate the TFS with some directories or files.

10.23 FILESYSTYPE statement



```
/*****************************************************************
  FILESYSTYPE TYPE(HFS)
            ENTRYPOINT(GFUAINIT)
            PARM(' ')
/*****************************************************************
  FILESYSTYPE TYPE(AUTOMNT)
            ENTRYPOINT(BPXTAMD)
/*****************************************************************/
  FILESYSTYPE TYPE(TFS)
            ENTRYPOINT(BPXTFS)
/*****************************************************************/
```

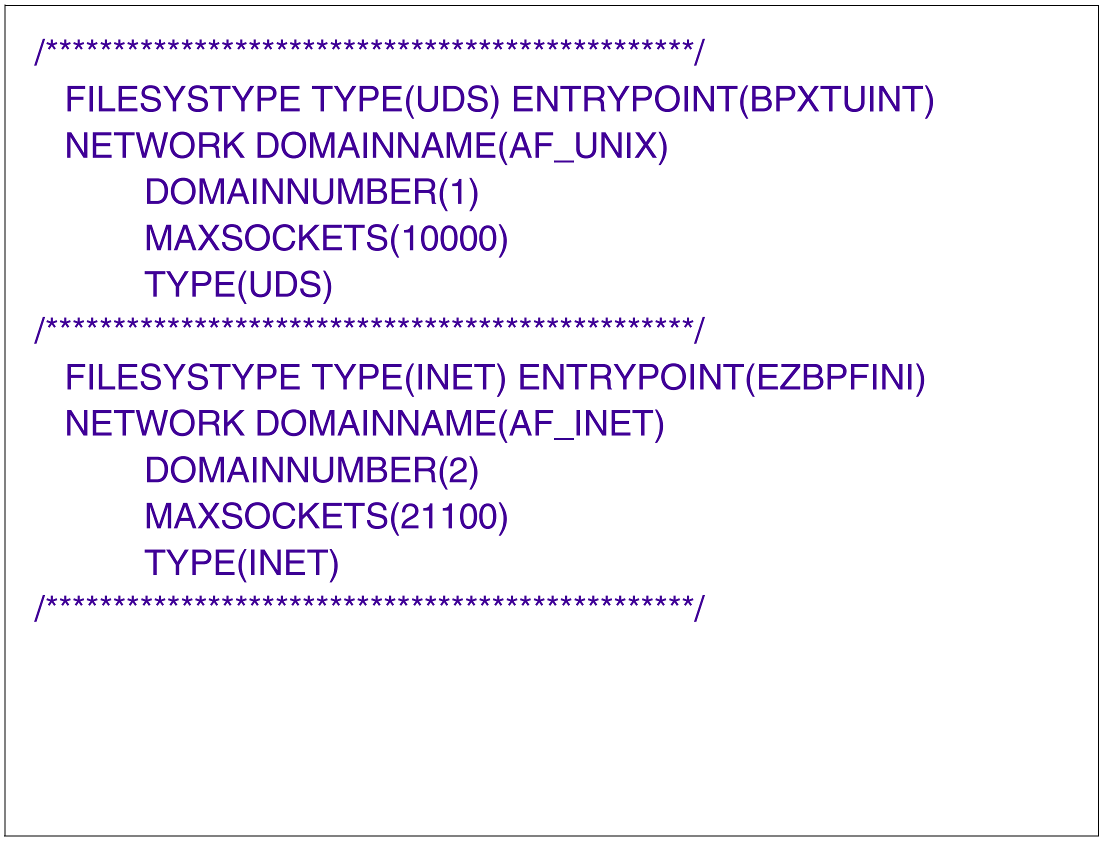
Figure 10-23 The FILESYSTYPE statement in the BPXPRMxx member

The following sections explain where the statements FILESYSTYPE, ROOT, MOUNT, and NETWORK apply.

FILESYSTYPE specifies that:

- A file system program of type HFS is to process file system requests. The system attaches the GFUAINIT load module during z/OS UNIX initialization.
- A physical file system of type AUTOMNT is to handle automatic mounting and unmounting of file systems. The system attaches the BPXTAMD load module during z/OS UNIX initialization.
- A physical file system of type TFS is to handle requests to the temporary file system. The system attaches the BPXTFS load module during z/OS UNIX initialization.

10.24 FILESYSTYPE and NETWORK



```
/*************************************************/
  FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)
  NETWORK DOMAINNAME(AF_UNIX)
        DOMAINNUMBER(1)
        MAXSOCKETS(10000)
        TYPE(UDS)
/*************************************************/
  FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
  NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(21100)
        TYPE(INET)
/*************************************************/
```

Figure 10-24 The FILESYSTYPE and NETWORK keywords

A physical file system of type UDS is to handle socket requests for the AF_UNIX address family of sockets. The system attaches the BPXTUINT load module during z/OS UNIX initialization.

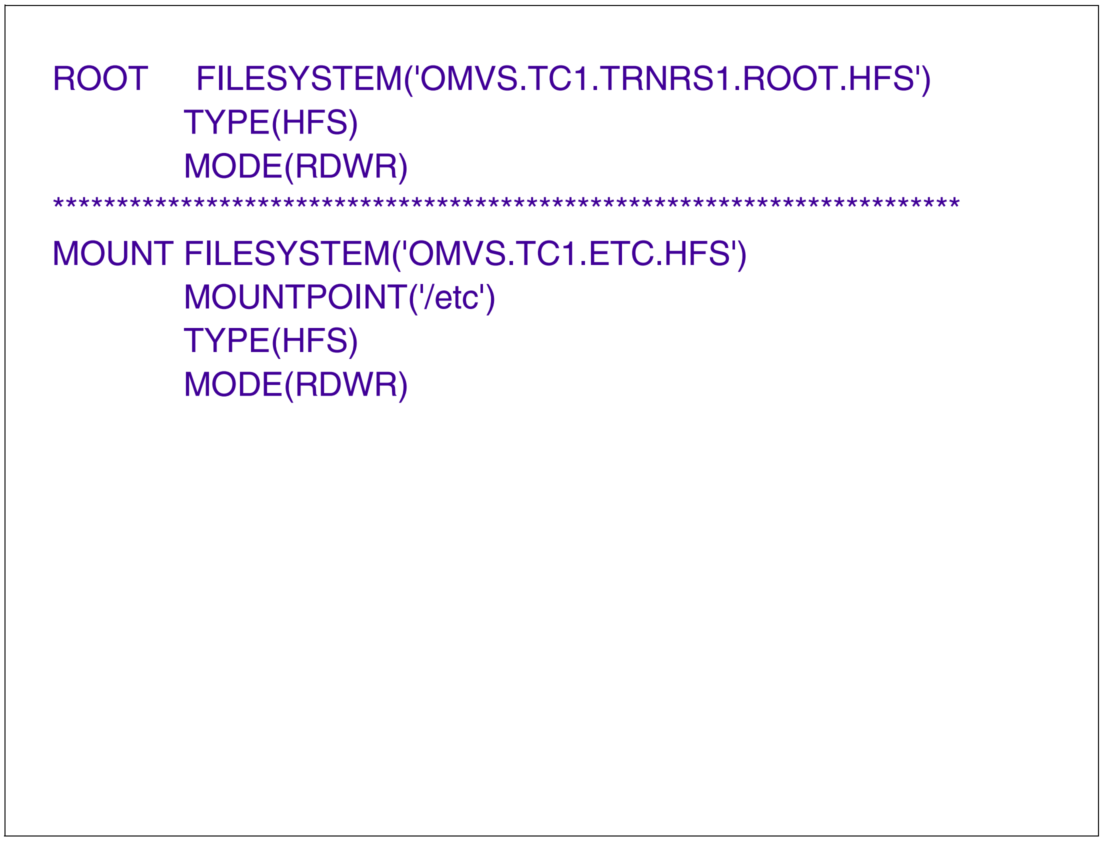
A physical file system of type INET is to handle requests for the AF_INET address family of sockets. The system attaches the EZBPFINI load module during initialization to start the AF_INET physical file system. This assumes you are using TCP/IP z/OS UNIX. If you want to use this, uncomment out the appropriate line in the BPXPRMxx parmlib member.

For more information about this, see *z/OS UNIX System Services Planning,* GA22-7800.

NETWORK - Each NETWORK statement identifies the information needed by the socket physical file system.

The AF_UNIX statement is used to communicate in the local system and AF_INIT to communicate with remote systems.

10.25 ROOT and MOUNT statements



```
ROOT     FILESYSTEM('OMVS.TC1.TRNRS1.ROOT.HFS')
         TYPE(HFS)
         MODE(RDWR)
***********************************************************************
MOUNT FILESYSTEM('OMVS.TC1.ETC.HFS')
         MOUNTPOINT('/etc')
         TYPE(HFS)
         MODE(RDWR)
```

Figure 10-25 The ROOT and MOUNT statements in the BPXPRMxx member

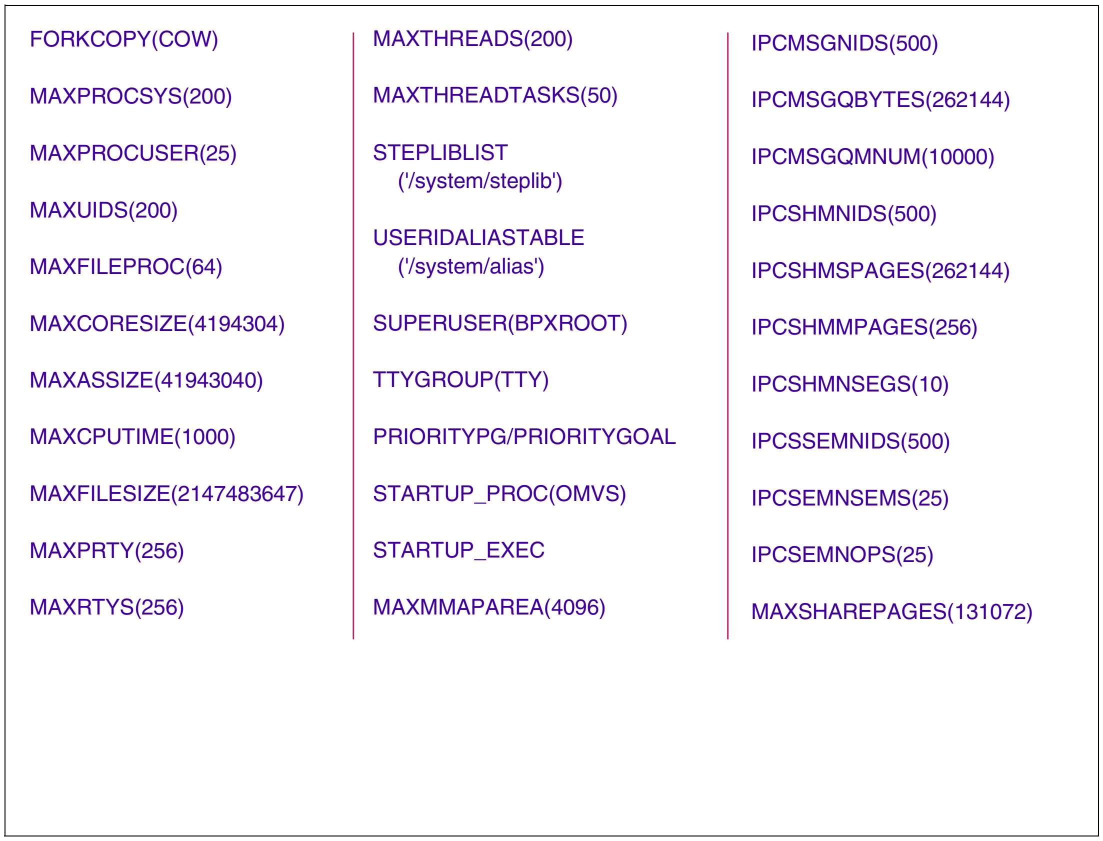
- **ROOT** identifies and mounts the HFS data set to be used as the root file system.
  - **TYPE** identifies the file system program, which must have been specified on a FILESYSTYPE statement.
  - MODE(RDWR) allows read and write access to the file system.

  In the ROOT statement, give the name of the root file system. Figure 10-25 shows:

  ```
  OMVS.TC1.TRNRS1.ROOT.HFS
  ```

- **MOUNT** - This statement provides a MOUNT statement for each HFS data set to be mounted on the root file system or on another mounted file system. The pictured example shows the data set `OMVS.TC1.ETC.HFS` is to be mounted at directory /etc.

10.26 BPXPRMxx summary


FORKCOPY(COW)

MAXPROCSYS(200)

MAXPROCUSER(25)

MAXUIDS(200)

MAXFILEPROC(64)

MAXCORESIZE(4194304)

MAXASSIZE(41943040)

MAXCPUTIME(1000)

MAXFILESIZE(2147483647)

MAXPRTY(256)

MAXRTYS(256)

MAXTHREADS(200)

MAXTHREADTASKS(50)

STEPLIBLIST
('/system/steplib')

USERIDALIASTABLE
('/system/alias')

SUPERUSER(BPXROOT)

TTYGROUP(TTY)

PRIORITYPG/PRIORITYGOAL

STARTUP_PROC(OMVS)

STARTUP_EXEC

MAXMMAPAREA(4096)

IPCMSGNIDS(500)

IPCMSGQBYTES(262144)

IPCMSGQMNUM(10000)

IPCSHMNIDS(500)

IPCSHMSPAGES(262144)

IPCSHMMPAGES(256)

IPCSHMNSEGS(10)

IPCSSEMNIDS(500)

IPCSEMNSEMS(25)

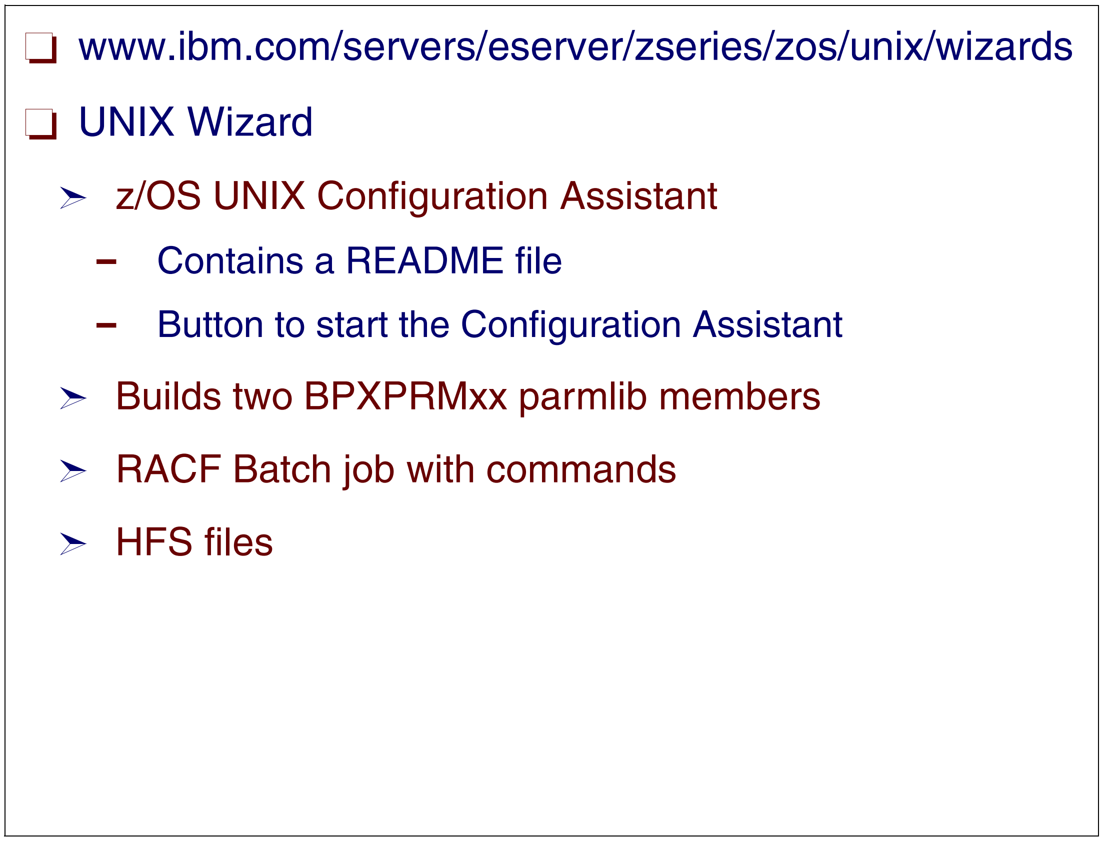
IPCSEMNOPS(25)

MAXSHAREPAGES(131072)

Figure 10-26 A summary of the BPXPRMxx statements

The BPXPRMxx statements we have gone through are the ones that can control resources used by z/OS UNIX processes, and threads within processes. Figure 10-26 provides a summary and overview of these statements.

10.27 z/OS UNIX Web site



- www.ibm.com/servers/eserver/zseries/zos/unix/wizards
- UNIX Wizard
  - z/OS UNIX Configuration Assistant
    - Contains a README file
    - Button to start the Configuration Assistant
  - Builds two BPXPRMxx parmlib members
  - RACF Batch job with commands
  - HFS files

Figure 10-27 Contents of the z/OS UNIX web site

Wizards are interactive assistants that ask you a series of questions about the task you want to do.

The z/OS UNIX configuration assistant is intended for those who are installing UNIX System Services for the first time. This wizard builds two BPXPRMxx parmlib members, a batch job with RACF commands to set up an initial security environment for testing z/OS UNIX, and several HFS files used in UNIX system processing. The BPXPRMxx parmlib members are tailored to your application environment. Current z/OS UNIX System Services users can use the tool to verify settings.

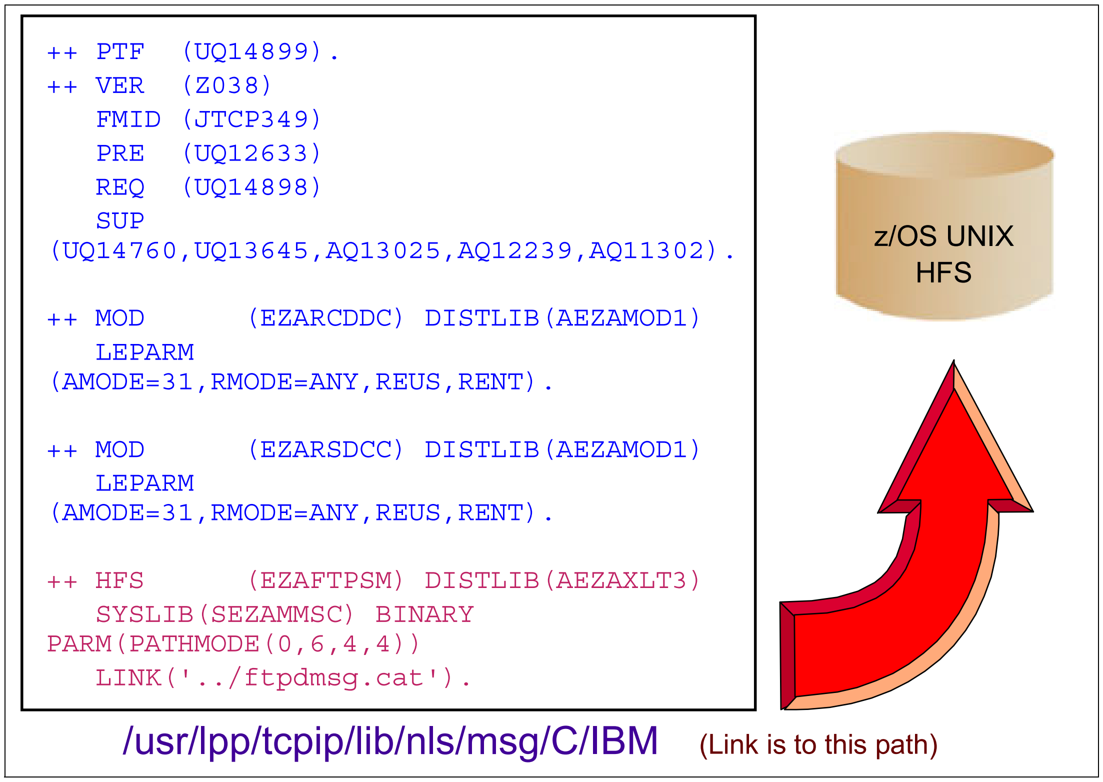
11

Maintenance

This chapter provides information about installing maintenance software with components residing in the z/OS UNIX file system. This chapter discusses the following topics:

- Overview of maintenance issues
- The HFS structure for SMP/E
- The SMP/E DDDEF and MOUNT relationship
- A method to support multiple service levels

11.1 Example SMP/E SMPMCS



```
++ PTF   (UQ14899).
++ VER   (Z038)
   FMID  (JTCP349)
   PRE   (UQ12633)
   REQ   (UQ14898)
   SUP
(UQ14760,UQ13645,AQ13025,AQ12239,AQ11302).

++ MOD       (EZARCDDC) DISTLIB(AEZAMOD1)
   LEPARM
(AMODE=31,RMODE=ANY,REUS,RENT).

++ MOD       (EZARSDCC) DISTLIB(AEZAMOD1)
   LEPARM
(AMODE=31,RMODE=ANY,REUS,RENT).

++ HFS       (EZAFTPSM) DISTLIB(AEZAXLT3)
   SYSLIB(SEZAMMSC) BINARY
PARM(PATHMODE(0,6,4,4))
   LINK('../ftpdmsg.cat').
```

Figure 11-1 Example of a SMP/E SMPMCS

z/OS now includes UNIX components as a standard part of the operating system. Therefore, it is now mandatory to have z/OS UNIX System Services enabled on any system where SMP/E maintenance is performed. Other traditional MVS products also now contain UNIX components; DB2 and NetView® are two such examples.

Figure 11-1 shows a typical PTF for TCP/IP (a component of z/OS). You will notice that it contains typical MCS statements such as `++ PTF`, `++ VER`, and `++ MOD`. You will also notice the `++ HFS` MCS statement, which directs element `EZAFTPSM` as a file into the HFS path pointed to by DDDEF `SEZAMMSC`. In addition, a *hard link* (alias name) will be created to `EZAFTPSM` in the directory one higher in the hierarchy (indicated by the ".." in the LINK parameter), called `ftpdmsg.cat`.

In other words, if the path in DDDEF SEZAMMSC points to:

```
/usr/lpp/tcpip/lib/nls/msg/C/IBM
```

Then the element will be written as file:

```
/usr/lpp/tcpip/lib/nls/msg/C/IBM/EZAFTPSM
```

With a hard link defined as:

```
/usr/lpp/tcpip/lib/nls/msg/C/ftpdmsg.cat
```

Note how the original file (`EZAFTPSM`) is written in the /C/IBM subdirectory, while the hard link (`ftpdmsg.cat`) is created in the /C subdirectory, one directory higher because of the ".." specification in the LINK parameter. This type of packaging allows SMP/E unique element names while also meeting the requirements of UNIX naming conventions.

11.2 Active root file system

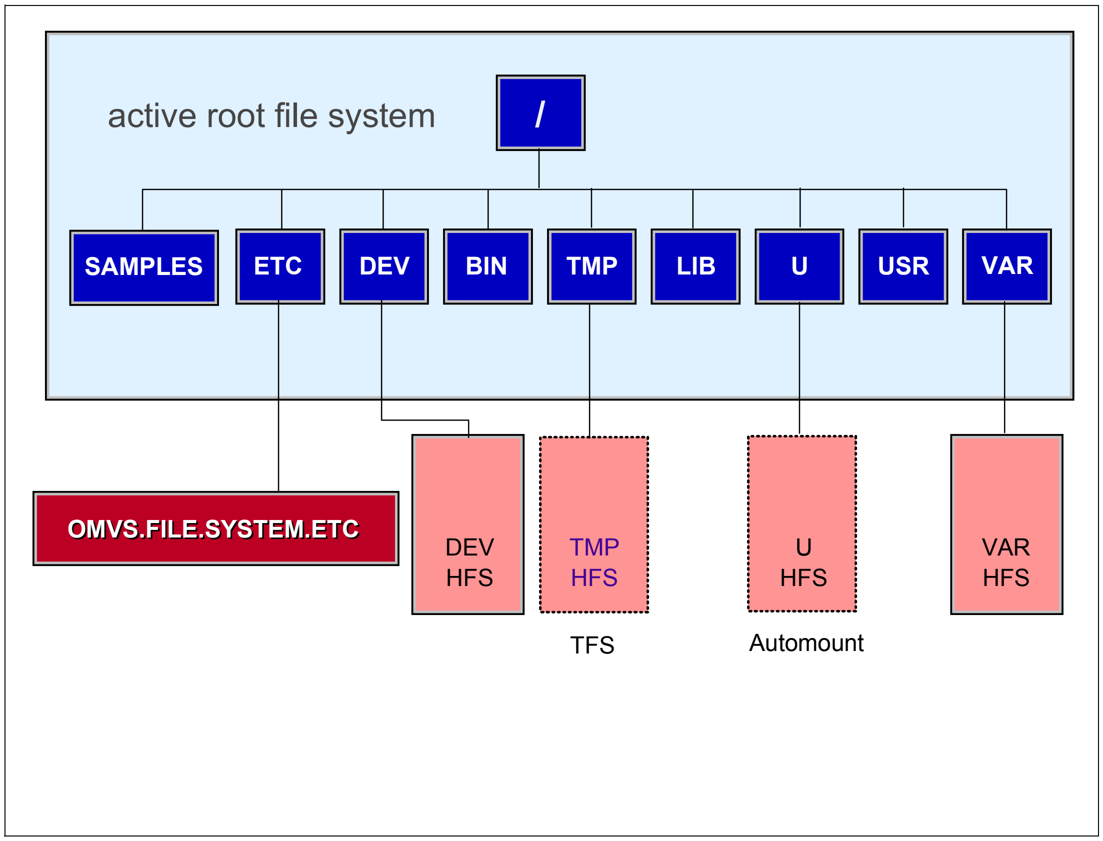


Figure 11-2 Example of an active root file system

Figure 11-2 is an example of an active file system. The file system is in use by the active system (the system you are logged on to), so applying maintenance directly onto the active file system is undesirable because:

- It could introduce a change which impacts work already running.
- If a problem is encountered during the SMP/E APPLY causing the APPLY to fail, it could damage the active file system and impact work already running.
- If you need to fall back to the point before the service was applied, the process is greatly complicated when it is the active file system.

The problems are similar to those concerning z/OS system residence (SYSRES) volumes, where typically a cloned copy of the active SYSRES is used to receive maintenance, then it is IPLed. This way, application of maintenance does not affect the active system until an IPL is performed. If there is a problem with the new maintenance level, fallback is to re-IPL from the old SYSRES.

11.3 Inactive root file system

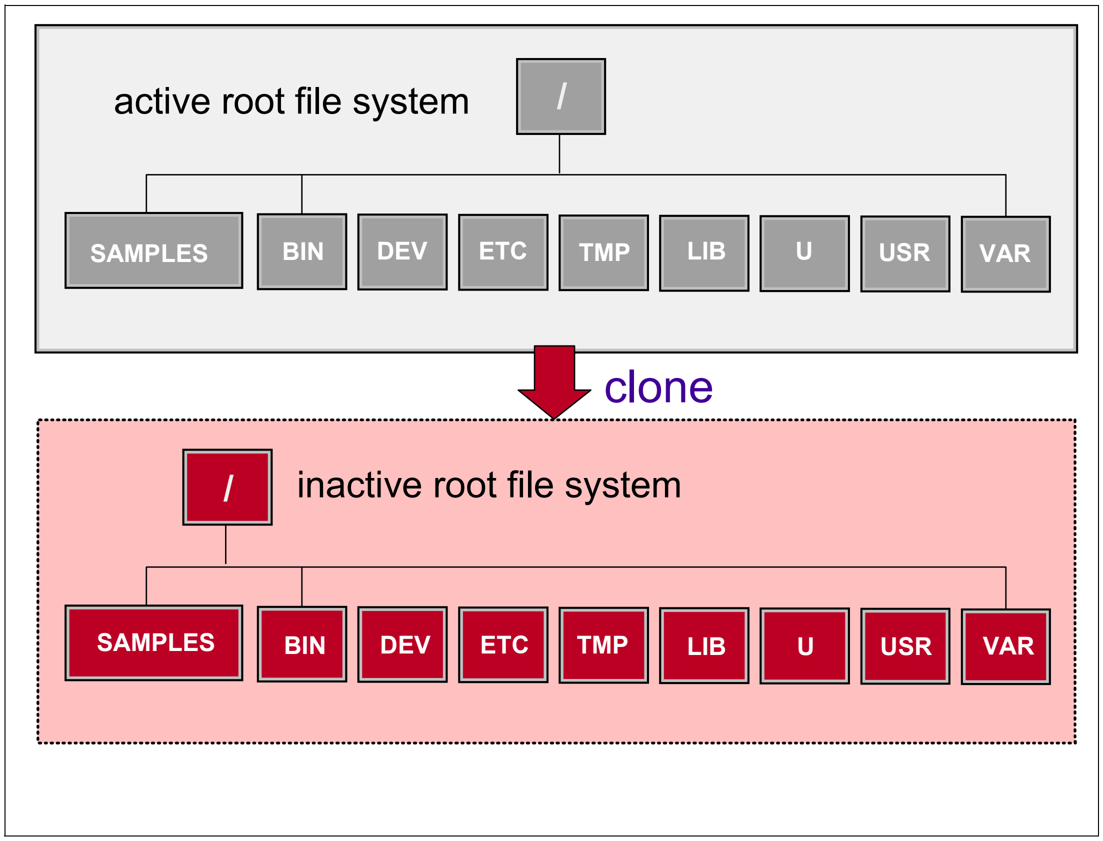


Figure 11-3 Making a clone of the active root file system

To apply maintenance safely without impact to an active system, we need to clone the z/OS UNIX file system, and work with the inactive copy of the file system. This is the same technique used for z/OS system residence (SYSRES) volumes, only the method has to vary because we are only dealing with HFS data sets and not full volumes.

To create a cloned copy of an HFS, DFSMSdss (ADRDSSU) must be used to first DUMP, then RESTORE with the RENAMEU (RENUNC) parameter, to result in a copied file with a different name.

Note: The DFSMSdss COPY function is supported for HFS data sets beginning in z/OS V1R3.

An example of the ADRDSSU JCL required to perform HFS cloning is shown in Example 11-1 on page 455.

Example 11-1 Sample JCL to perform HFS cloning

```
//DUMPREST JOB ,'DUMP/REST',CLASS=A
//DUMP     EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//OUTDD    DD DISP=(NEW,PASS),DSN=&&TEMP,
//            UNIT=SYSALLDA,
//            SPACE=(TRK,(100,100))
//SYSIN    DD *
 DUMP DATASET( INCLUDE( OMVS.RESA01.** ) ) -
OUTDD(OUTDD) -
       CANCELERROR TOL(ENQF)
/*
//*
//*-------------------------------------------------------------------
//*
//RESTORE  EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//INDD     DD DISP=(OLD,DELETE),DSN=&&TEMP
//SYSIN    DD *
RESTORE DATASET( -
           INCLUDE( - ** ) ) -
          INDD(INDD) -
      RENUNC( (OMVS.RESA01.**, OMVS.RESB01.**) ) -
          TGTALLOC(SOURCE) -
          TOL(ENQF)
/*
```

11.4 /SERVICE directory

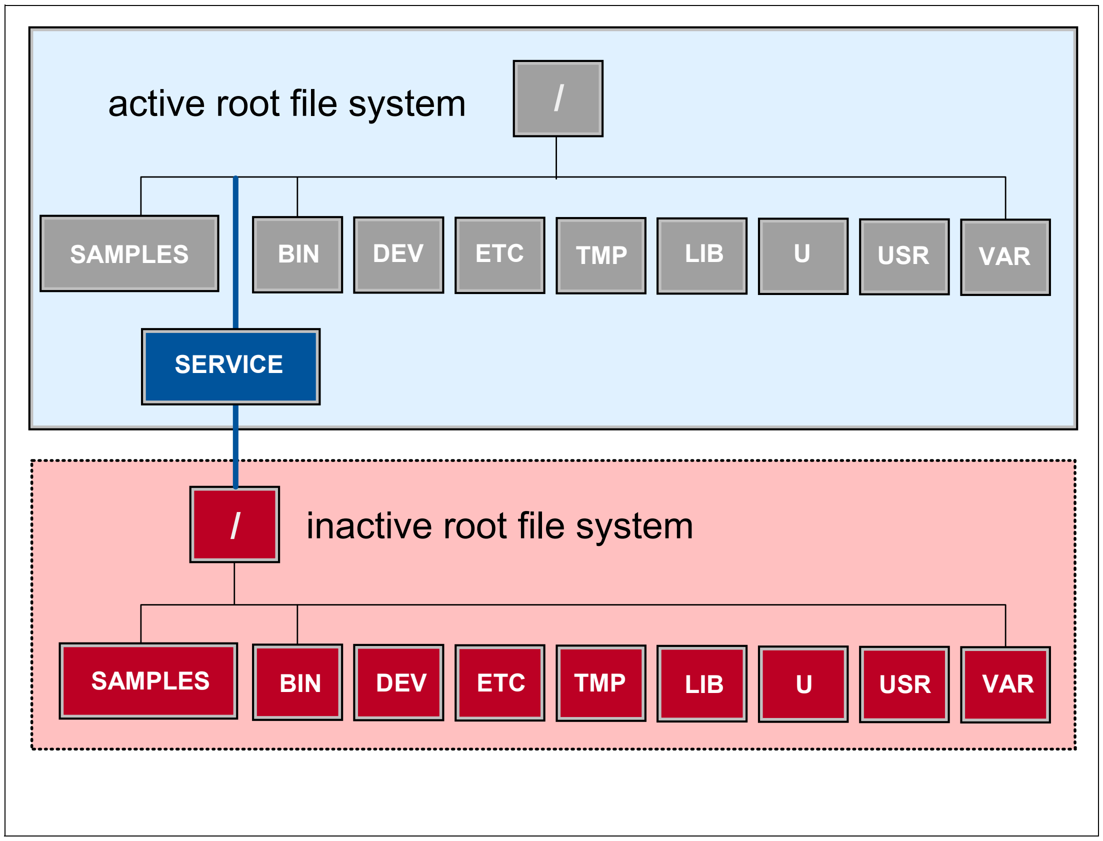


Figure 11-4 Creating a /SERVICE directory

z/OS UNIX can only access files if the HFS that contains them is mounted into the active file system. So, the newly created clone file system needs to be somehow connected into the active file system structure so SMP/E processes can access the data.

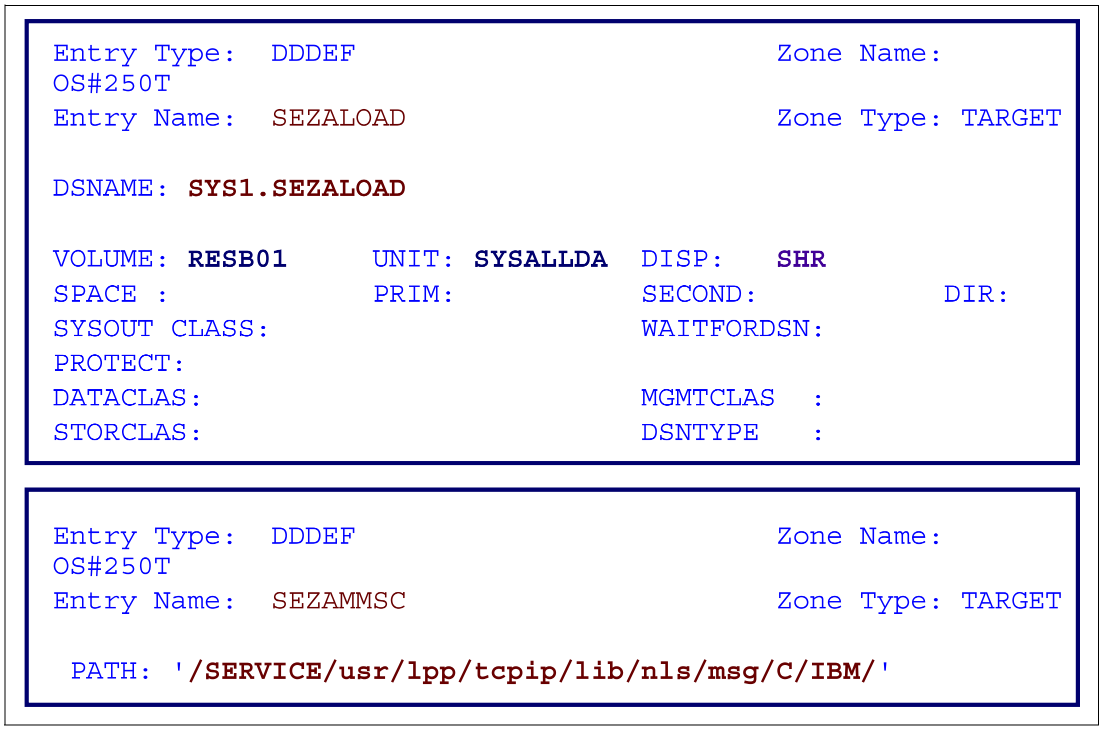
The file system could be connected anywhere in the active file system, but the recommended place to do it is under a directory called /SERVICE. Using this directory will ensure no impact to other UNIX processing.

If this directory does not exist on your system, you could create it via a TSO command, shell command, or using the ISHELL. As an example, the TSO command to do this is:

```
MKDIR '/SERVICE' MODE(7,5,5)
```

Once the /SERVICE directory has been created, the cloned file system needs to be mounted there. The mount can happen by specifying the HFS in BPXPRMxx or issuing a TSO **MOUNT** command.

11.5 Example SMP/E DDDEFs



```
Entry Type:  DDDEF                                Zone Name:
OS#250T
Entry Name:  SEZALOAD                             Zone Type: TARGET

DSNAME: SYS1.SEZALOAD

VOLUME: RESB01     UNIT: SYSALLDA  DISP:   SHR
SPACE :            PRIM:           SECOND:             DIR:
SYSOUT CLASS:                      WAITFORDSN:
PROTECT:
DATACLAS:                          MGMTCLAS   :
STORCLAS:                          DSNTYPE    :
```

```
Entry Type:  DDDEF                                Zone Name:
OS#250T
Entry Name:  SEZAMMSC                             Zone Type: TARGET

 PATH: '/SERVICE/usr/lpp/tcpip/lib/nls/msg/C/IBM/'
```

Figure 11-5 An example of a SMP/E DDDEFs

Once the HFS data sets are cloned into the /SERVICE directory, some decisions need to be made about the SMP/E configuration that will be used to support the cloned environment. You could either change the DDDEFs of the old SMP/E configuration (meaning that the active environment is no longer maintainable), or you could clone the SMP/E CSIs and change the new CSIs to point to the new HFS.

Although we are mainly talking about HFS data sets in this section, of course you must ensure the integrity of the SMP/E configuration and make sure that all DDDEFs in the CSIs point to data sets with contents that match what is recorded in SMP/E. How this is handled may be different depending on the product and how it is implemented on your system. We discuss this in more detail in the next few sections.

Regardless of how you choose to solve the problem of SMP/E integrity, you will still have to adjust the SMP/E DDDEFs to point to the cloned file system mounted at /SERVICE. The SMP/E **ZONEEDIT** command can be used to do this.

Figure 11-5 shows a traditional DDDEF (top box) and an HFS DDDEF (bottom box). The traditional DDDEF SEZALOAD points to a PDS called `SYS1.SEZALOAD` on DASD volume `RESB01`, while the HFS DDDEF SEZAMMSC points to a path called /SERVICE/usr/lpp/tcpip/lib/nls/msg/C/IBM/ (note the /SERVICE directory).

11.6 Prepare for SMP/E

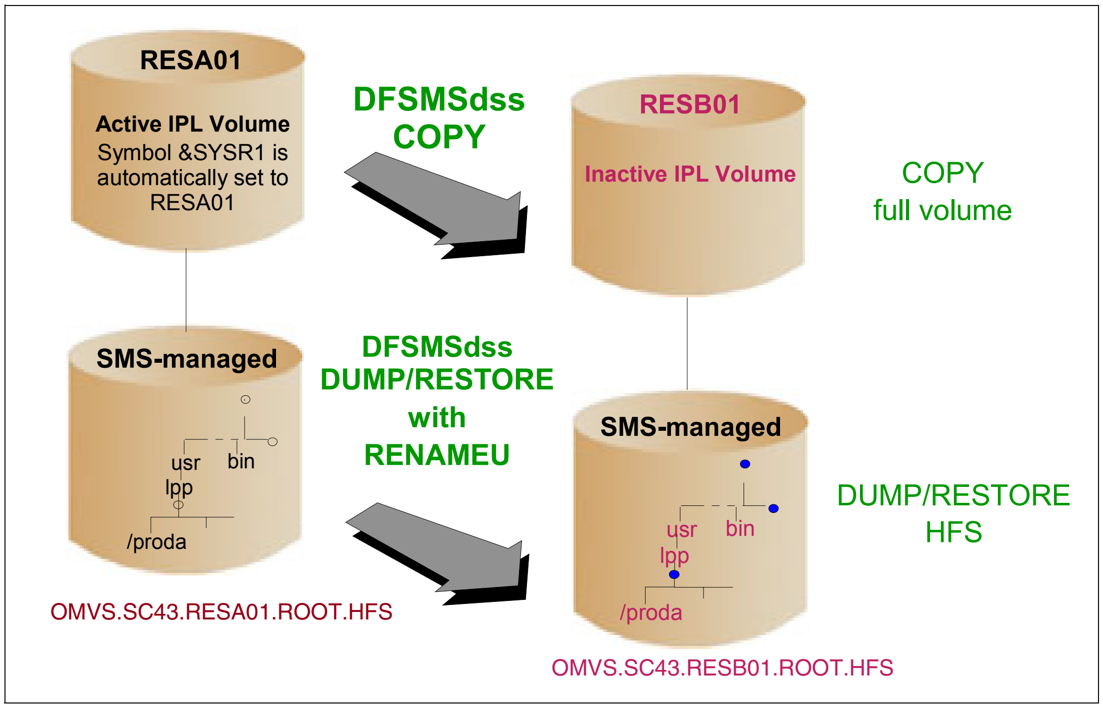


Figure 11-6 Preparing for the SMP/E maintenance

The preparation for system maintenance should include the following steps:

- Clone the data sets that the SMP/E DDDEFs point to:
  - For non-HFS data sets, you would DFSMSdss (ADRDSSU) COPY the active files to create inactive equivalents. You should consider that:
    - SYSRES data sets would have the entire volume copied, such that the inactive data sets have the same names but are uncataloged. This figure shows active volume RESA01 being copied to RESB01.
    - Non-SYSRES data sets may either be copied to another volume with the same names, but uncataloged; or they may be copied with other names, allowing them to be cataloged if you choose.
  - For HFS data sets, you would DFSMSdss (ADRDSSU) DUMP/RESTORE the active files to create inactive equivalents (as described in previous sections). You should consider that:
    - HFS data sets that belong to SYSRES products need something in their name to associate them to the correct SYSRES. For example, say you clone SYSRES volume RESA01 onto volume RESB01: the HFS data sets that relate to RESB01 should have RESB01 in their name. This allows the use of the &SYSR1. symbol in the BPXPRMxx member so that the HFS data sets that relate to the IPLed SYSRES are always correctly selected.

      For example, BPXPRMxx contained a reference to HFS data set OMVS.&SYSNAME..&SYSR1..ROOT.HFS.
    - If you IPLed from volume RESA01, symbol &SYSR1. would be substituted with string RESA01 resulting in a data set name of OMVS.SC43.RESA01.ROOT.HFS.

    - If you IPLed from volume RESB01, symbol &SYSR1. would be substituted with string RESB01 resulting in a data set name of OMVS.SC43.RESB01.ROOT.HFS.
  - This technique simplifies moving back and forth between maintenance levels (test sessions, implementations, and so on).
  - Because HFS data sets cannot be shared for write, HFS data set names in a shared DASD configuration may need a system identifier (&SYSNAME.) in them to differentiate which system they belong to. This may be more complicated in a shared DASD configuration where multiple systems are IPLed from the same DASD volume. In this situation, both &SYSNAME. and &SYSR1. may be needed in the affected HFS names. For example: OMVS.&SYSNAME..&SYSR1..ROOT.HFS
- Make SMP/E point to the inactive data. You could choose to:
  - Change the DDDEFs in the existing SMP/E CSIs to point to the inactive data. This would result in the active data having no SMP/E CSIs pointing to them, which may be a problem if an emergency fix needs to be applied to the live system.
  - Clone the SMP/E CSIs so the old CSIs continue pointing to the active data, while the DDDEFs in the new CSIs can be ZONEEDITed to point to the inactive data.

Note: The DFSMSdss COPY function is supported for HFS data sets beginning in z/OS V1R3.

11.7 SMP/E APPLY process

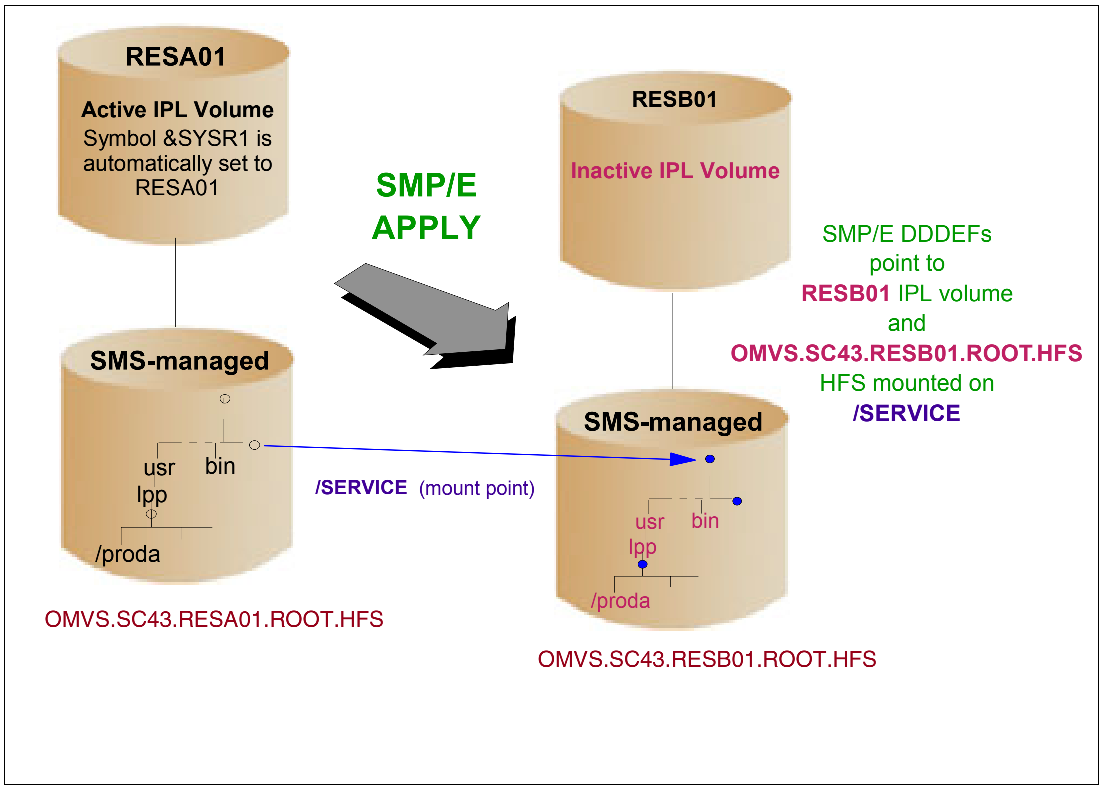

RESA01
Active IPL Volume
Symbol &SYSR1 is automatically set to RESA01
SMP/E APPLY
RESB01
Inactive IPL Volume
SMP/E DDDEFs point to RESB01 IPL volume and OMVS.SC43.RESB01.ROOT.HFS HFS mounted on /SERVICE
SMS-managed
usr
bin
lpp
/proda
/SERVICE (mount point)
SMS-managed
usr
bin
lpp
/proda
OMVS.SC43.RESA01.ROOT.HFS
OMVS.SC43.RESB01.ROOT.HFS

Figure 11-7 Doing the SMP/E APPLY

For the SMP/E APPLY process to work as desired, you should have inactive clones of the active data sets.

- HFS data sets should be mounted into the active file system off the /SERVICE directory. The data sets could be mounted there permanently via the BPXPRMxx member, or dynamically via a TSO **MOUNT** command. For the example shown in Figure 11-7, the following command could be used:

```
MOUNT FILESYSTEM('OMVS.SC43.RESB01.ROOT.HFS') +
    TYPE(HFS) MODE(RDWR) MOUNTPOINT('/SERVICE')
```

- If you wanted to dynamically remove the file system after maintenance work has been performed, the following command could be used:

```
UNMOUNT FILESYSTEM('OMVS.SC43.RESB01.ROOT.HFS') +
      IMMEDIATE
```

SMP/E CSIs point to the inactive data sets. No matter how you choose to set up your SMP/E CSIs as described in the previous section, you will probably need to use an SMP/E **ZONEEDIT** command such as the following to get the DDDEFs set correctly:

```
ZONEEDIT DDDEF.
      CHANGE VOLUME(RESA01,
                    RESB01).
      CHANGE PATH('/'*,
                  '/SERVICE/'*).
    ENDZONEEDIT.
```

11.8 Supporting multiple service levels

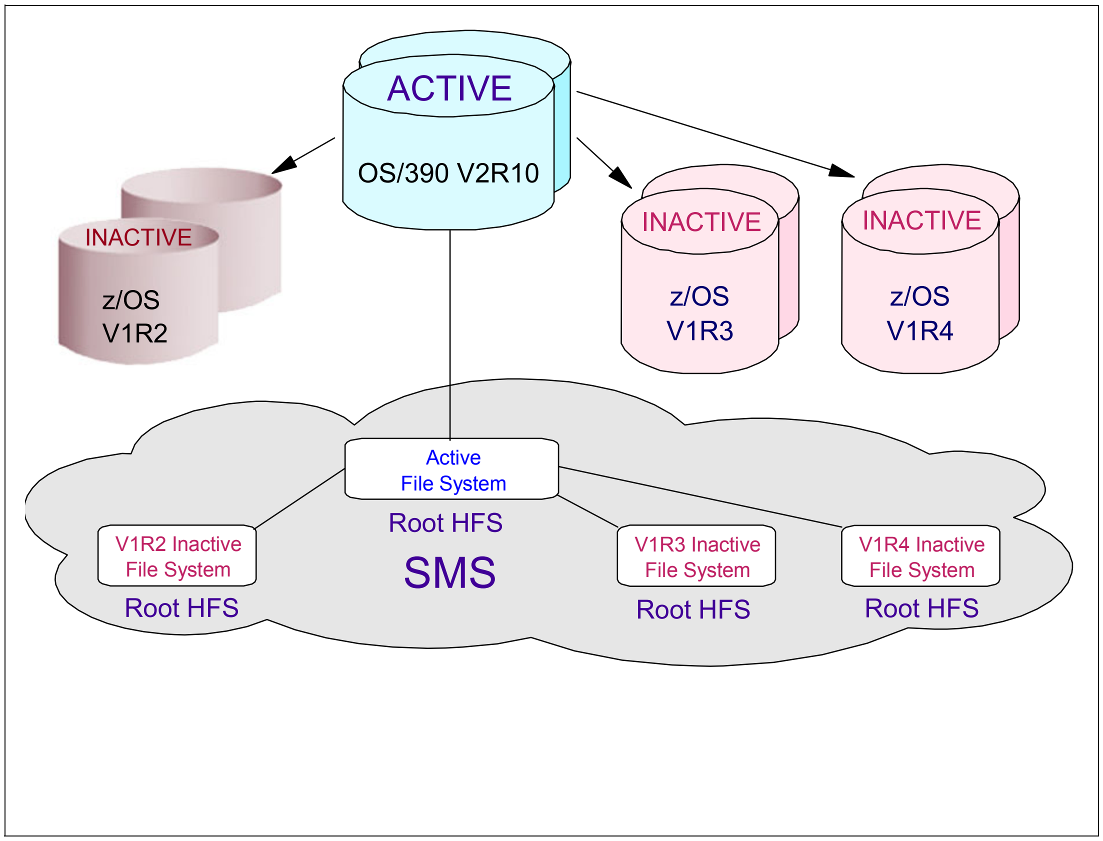


Figure 11-8 Supporting multiple service levels

Suppose you need to support more than one level of software as you migrate from one level to the next. Or, maybe you work in a large organization and have to support many levels of software simultaneously. For this, some additional considerations are necessary, particularly concerning the HFS data sets and file system structure.

Figure 11-8 shows an active OS/390 V2R10 system where the SMP/E work is performed to support z/OS V1R2, V1R3, and V1R4. You will notice there are multiple HFS data sets in SMS to match all the OS/390 and z/OS levels, and which need to be somehow connected into the file system. Until now we have only talked about using the /SERVICE directory to support one inactive file system for maintenance. We could still do this by mounting the correct file system when doing maintenance, then unmounting it afterwards, but there are better ways than that, as shown in Figure 11-9 on page 462.

11.9 Supporting multiple service levels (2)

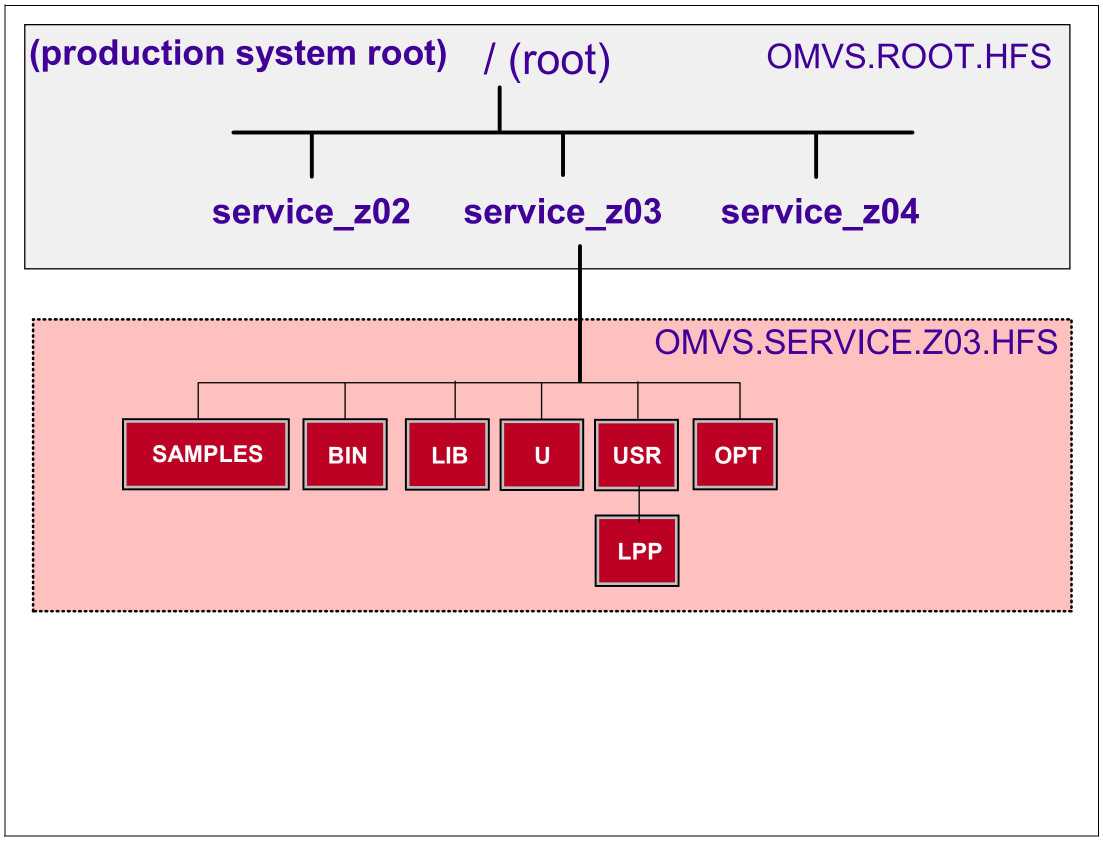


Figure 11-9 A directory structure for multiple service levels

This figure shows a possible directory structure for connecting multiple service levels into the **/** directory. The inactive file systems would be mounted something like:

- z/OS V1R2 at **/service_z02**
- z/OS V1R3 at **/service_z03**
- z/OS V1R4 at **/service_z04**

There is no problem in doing this as long as the path in the DDDEFs match the directory structure. So, an SMP/E **ZONEEDIT** command something like the following may be required to set the SMP/E DDDEFs correctly:

```
ZONEEDIT DDDEF.
CHANGE PATH('/'*, '/service_z03’).
ENDZONEEDIT.
```

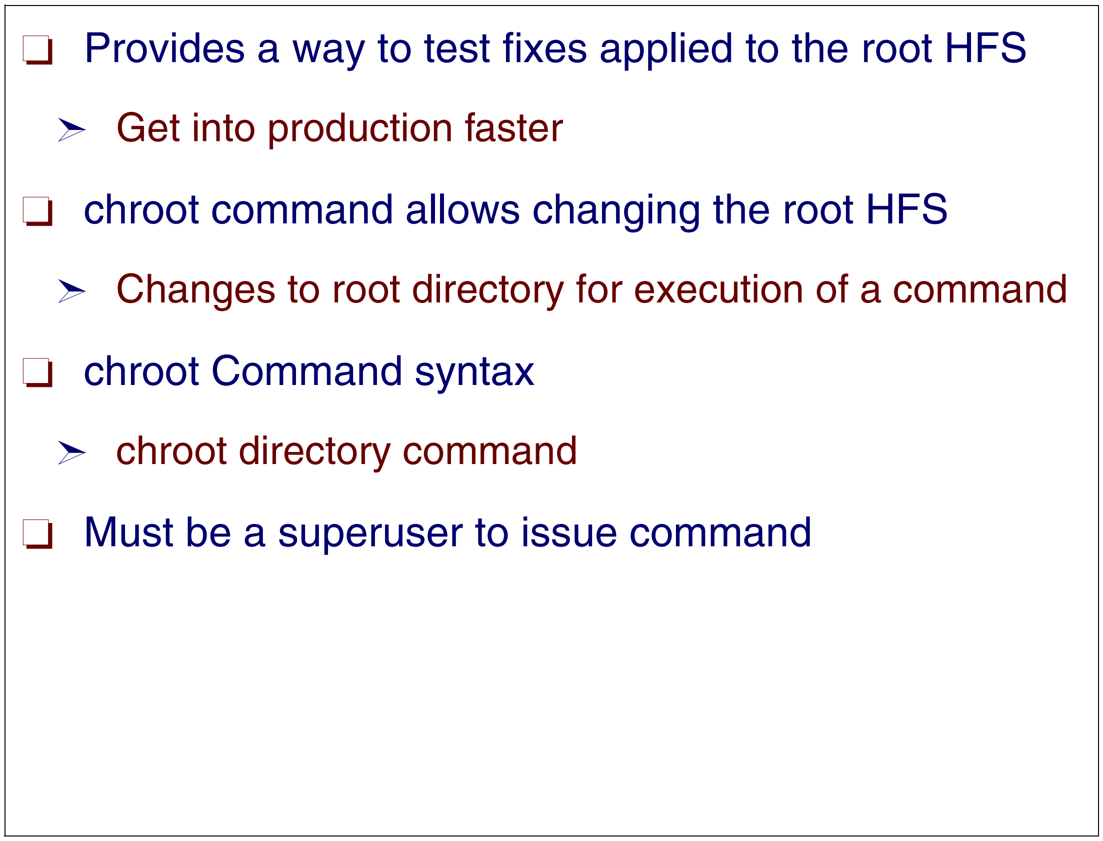
11.10 ISHELL display of root

```
   Type  Filename
Row 1 of 32
 _ Dir   .
 _ Dir   ..
 _ File  .sh_history
 _ Dir   bin
 _ Syml  dev
 _ Syml  etc
 _ Dir   imsjava
 _ Syml  krb5
 _ Dir   lib
 _ Dir   opt
 _ Dir   samples
 _ Dir   service_db2v6
 _ Dir   service_hpj
 _ Dir   service_imsv7
 _ Dir   service_netview13
 _ Dir   service_netvu130
 _ Dir   service_z02
 _ Dir   service_z03
 _ Dir   service_z04
```

Figure 11-10 An ISHELL display of the root directory

This figure shows the ITSO system where maintenance is done. It shows the directories that were added to the Root for each of the system levels of z/OS supported and the individual products where maintenance is applied.

11.11 chroot command



- Provides a way to test fixes applied to the root HFS
  - Get into production faster
- chroot command allows changing the root HFS
  - Changes to root directory for execution of a command
- chroot Command syntax
  - chroot directory command
- Must be a superuser to issue command

Figure 11-11 Using the chroot command

By using the /SERVICE directory scheme, many customers are maintaining multiple releases of z/OS UNIX from one driving system. The **chroot** command will allow the system programmer to test out fixes and new releases easier and faster.

This command is not part of the XPG4 or UNIX98 standard. The externals are somewhat modeled on AIX externals.

The directory path name is always relative to the current root. After **chroot** is issued, your current working directory is the new root (directory). **chroot** does not change environment variables.

In order for the command to operate properly after the **chroot** is issued, you need to have all the files in the new root directory that the command depends on.

Note: You must be a superuser to issue the **chroot** command.

11.12 Testing a root file system

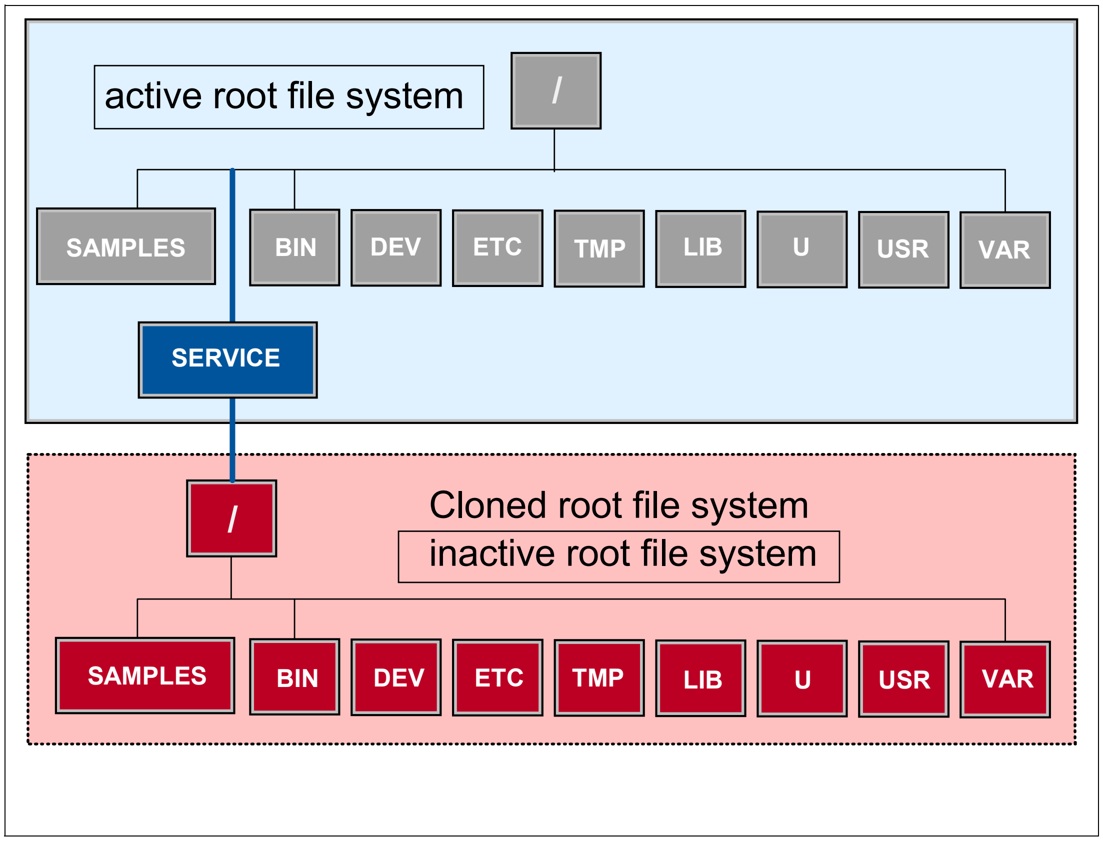


Figure 11-12 Testing the updated clone of the root file system

Figure 11-12l shows the new inactive root file system mounted on the /SERVICE directory in the active root file system.

11.13 Testing the updated root

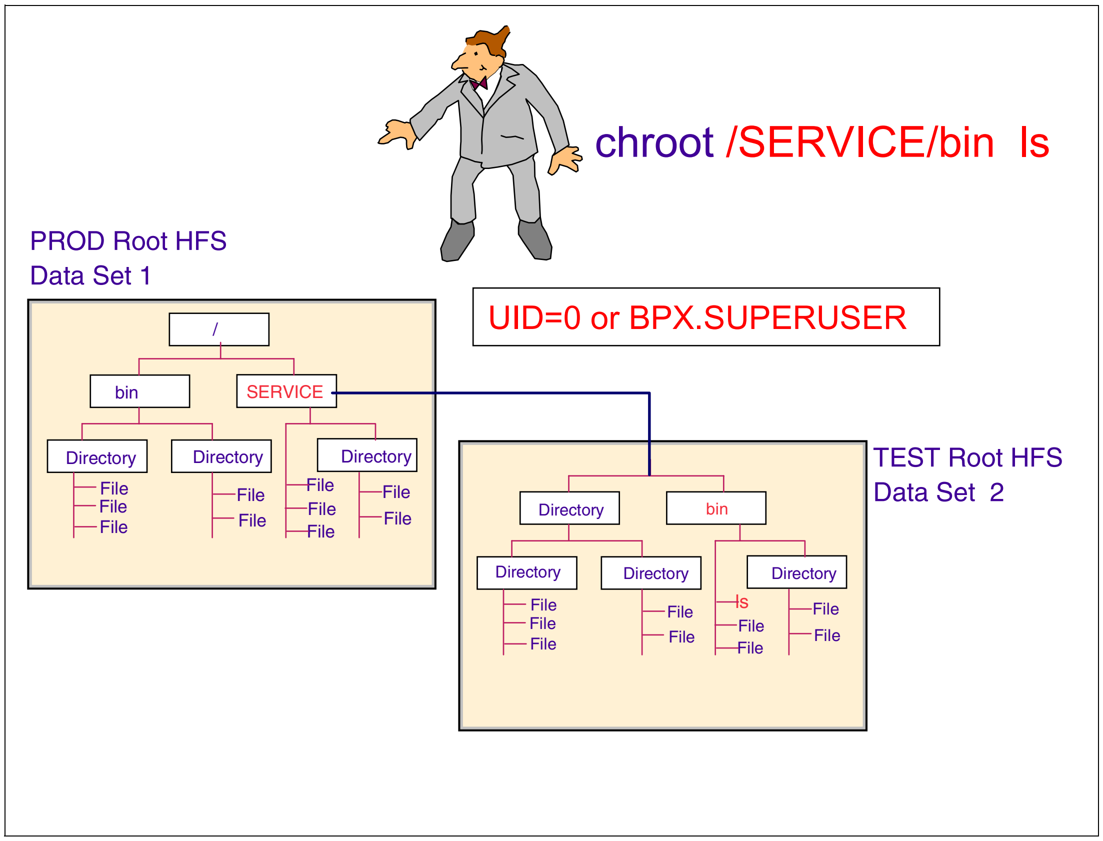


Figure 11-13 Testing the updated root with the chroot command

If you have appropriate privileges, the **chroot** command changes the root directory to the directory specified by the directory parameter of a specific command. The new root directory will also contain its children.

In order to use **chroot**, you must either be a superuser (UID=0), or be a member of the BPX.SUPERUSER facility class.

The directory path name is always relative to the current root. If a nested **chroot** command is in effect, the directory path name is still relative to the current (new) root of the running process.

In order for your process to operate properly after the **chroot** is issued you need to have in your new root all the files that your programs depend on. For example, if your new root is /SERVICE and you issue an **ls**, you will get a not found error. To use **ls** with /SERVICE as your new root, you will need a /SERVICE/bin with `ls` in it before you issue the **chroot** command.

After **chroot** is issued, your current working directory is the new root (directory); chroot does not change environment variables.

z/OS UNIX operations

This chapter provides an overview of UNIX System Services operations. It provides details about:

- z/OS UNIX operator commands
- z/OS UNIX Abends and Messages
- Sources of debugging information for z/OS UNIX

12.1 Commands to monitor z/OS UNIX

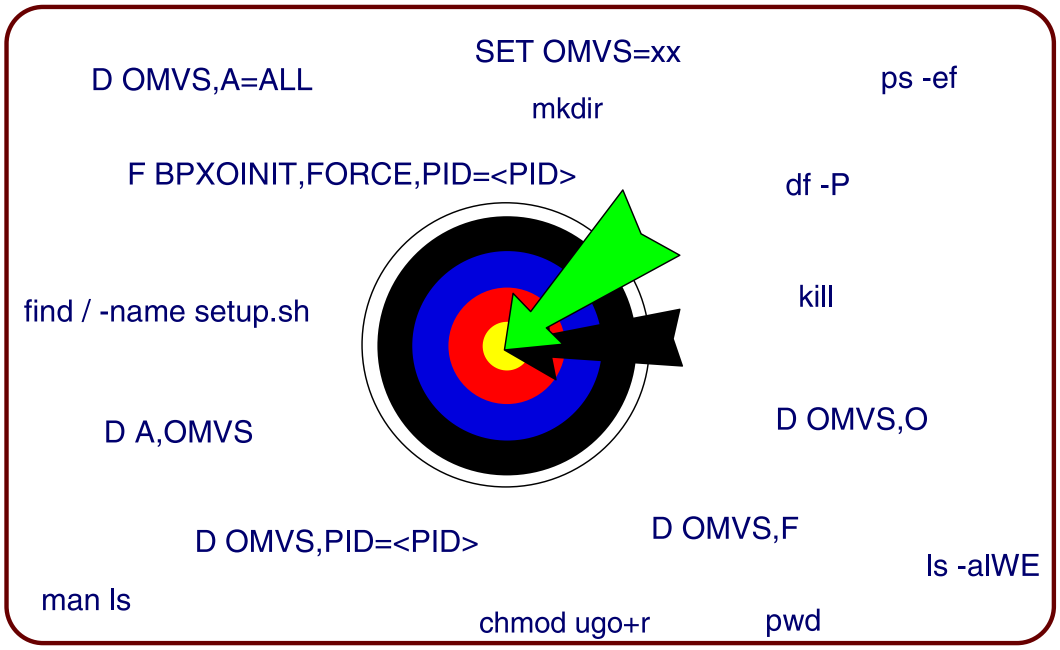


Figure 12-1 Commands used to monitor z/OS UNIX

There are many different reasons to monitor z/OS UNIX, for example:

- To kill a process
- To see how much space a directory needs
- To activate a new configuration for z/OS UNIX
- To find a file located in the HFS
- To change your local directory

Therefore, you need to know which commands show the information that you want, and which commands provide useful information without incurring too much overhead.

Commands can be divided into two sections: commands you issue from a console, and commands you issue from the z/OS UNIX shell.

z/OS console commands

`D OMVS,A=ALL`	Displays the information about all running processes
`D OMVS,O`	Displays the information defined in the BPXPRMxx
`D OMVS,F`	Displays the information about the mounted HFS
`D OMVS,PID=<pid>`	Displays the information about the process ID
`D A,OMVS`	Displays the information about kernel and dataspaces
`F BPXOINIT,FORCE,PID=<PID>`	Forces a process ID to end
`SET OMVS=xx`	Sets a new configuration of BPXPRMxx member and makes a syntax check

z/OS UNIX shell commands

`pwd`	Displays the current pathname
`ls -alWE`	Displays the contents and extended attributes of the current directory
`ps -ef`	Displays the information about all running processes
`man ls`	Displays information about syntax and use of the command **`ls`**
`df -P`	Displays the information about the mounted HFS
`find / -name setup.sh`	Searches the HFS from the root to find the file specified

12.2 Display summary of z/OS UNIX

```
D OMVS
BPXO042I 13.31.39 DISPLAY OMVS 908
OMVS      000E ACTIVE          OMVS=(00,FS)
```

Figure 12-2 Command to display the status of z/OS UNIX

The **D OMVS** command with no parameters displays the status of z/OS UNIX (hopefully always `ACTIVE`), and which BPXPRMxx member was defined during IPL. The value `000E` shown in Figure 12-2 is the address space identifier (ASID) of the kernel address space.

The status of z/OS UNIX can be:

ACTIVE	z/OS UNIX is currently active
NOT STARTED	z/OS UNIX was not started
INITIALIZING	z/OS UNIX is initializing
TERMINATING	z/OS UNIX is terminating
TERMINATED	z/OS UNIX has terminated
ETC/INIT WAIT	z/OS UNIX is waiting for the /etc/init or /usr/sbin/init program to complete initialization

12.3 Display z/OS UNIX options

```
D,OMVS,OPTIONS
BPXO043I 13.21.39 DISPLAY OMVS 923
OMVS     000E ACTIVE          OMVS=(3C)
CURRENT UNIX CONFIGURATION SETTINGS:
MAXPROCSYS      =       4096    MAXPROCUSER     =      32767
MAXFILEPROC     =      65535    MAXFILESIZE     = NOLIMIT
MAXCPUTIME      = 2147483647    MAXUIDS         =        200
MAXPTYS         =        800
MAXMMAPAREA     =       4096    MAXASSIZE       = 2147483647
MAXTHREADS      =     100000    MAXTHREADTASKS  =      32767
MAXCORESIZE     =    4194304    MAXSHAREPAGES   =     331072
IPCMSGQBYTES    =     262144    IPCMSGQMNUM     =      10000
IPCMSGNIDS      =      20000    IPCSEMNIDS      =      20000
IPCSEMNOPS      =      32767    IPCSEMNSEMS     =      32767
IPCSHMMPAGES    =     524287    IPCSHMNIDS      =      20000
IPCSHMNSEGS     =       1000    IPCSHMSPAGES    =    2621440
SUPERUSER       = BPXROOT       FORKCOPY        = COW
STEPLIBLIST     = /usr/lpp/IMiner/steplib
USERIDALIASTABLE= /etc/ualiastable
PRIORITYPG VALUES: NONE
PRIORITYGOAL VALUES: NONE
MAXQUEUEDSIGS   =     100000    SHRLIBRGNSIZE   =   67108864
SHRLIBMAXPAGES  =    3145728    VERSION         = Z03RD1
SYSCALL COUNTS  = NO            TTYGROUP        = TTY
SYSPLEX         = YES           BRLM SERVER     = SC04
LIMMSG          = SYSTEM        AUTOCVT         = OFF
RESOLVER PROC   = DEFAULT
```

Figure 12-3 Command to display the z/OS UNIX options defined in the BPXPRMxx member

The **`D OMVS,OPTIONS`** command displays the same information as the **`D OMVS`** command, but in addition, all the option settings that were defined in BPXPRMxx.

This command displays the current settings of the options that were:

- Set during initialization in the parmlib member BPXPRMxx
- Set by a **`SET OMVS`** or **`SETOMVS`** command after initialization, and that can be altered dynamically by these commands

You can use this command to display address space information for a user who has a process that is hung. You can also use the information returned from this command to determine how many address spaces a given TSO/E user ID is using, whether an address space is using too many resources, and whether a user's process is waiting for a z/OS UNIX kernel function to complete.

12.4 Display BPXPRMxx limits

```
D,OMVS,LIMITS
BPXO051I 17.50.39 DISPLAY OMVS 356
OMVS     000F ACTIVE        OMVS=(4A)
SYSTEM WIDE LIMITS:        LIMMSG=NONE
                  CURRENT  HIGHWATER     SYSTEM
                    USAGE      USAGE      LIMIT
MAXPROCSYS             50         51        300
MAXUIDS                 0          0         50
MAXPTYS                 0          0        256
MAXMMAPAREA          3572       3572       4096
MAXSHAREPAGES        4516       4516   32768000
IPCMSGNIDS             10         10      20000
IPCSEMNIDS              0          0      20000
IPCSHMNIDS              0          0      20000
IPCSHMSPAGES            0          0    2621440
IPCMSGQBYTES          ---          0     262144
IPCMSGQMNUM           ---          0      10000
IPCSHMMPAGES          ---          0      25600
SHRLIBRGNSIZE    11534336   11534336   67108864
SHRLIBMAXPAGES        472        472       4096
```

Figure 12-4 Command to display the limits specified in the BPXPRMxx member

Using the **D OMVS,LIMITS** command, you can display information about current system-wide parmlib limits, including current usage and high-water usage.

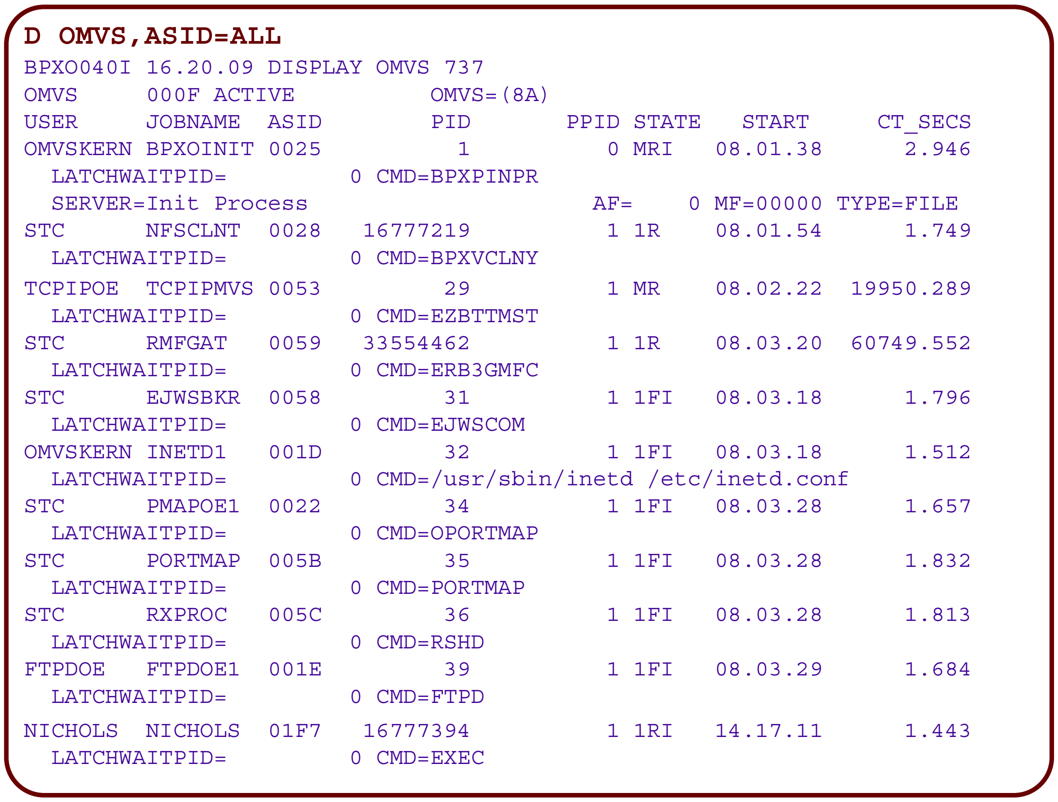
An asterisk (*) displayed after a system limit indicates that the system limit was changed via a **SETOMVS** or **SET OMVS** command.

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

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

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

12.5 Display address space information



```
D OMVS,ASID=ALL
BPXO040I 16.20.09 DISPLAY OMVS 737
OMVS     000F ACTIVE          OMVS=(8A)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 0025          1          0 MRI    08.01.38     2.946
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                    AF=    0 MF=00000 TYPE=FILE
STC      NFSCLNT  0028   16777219          1 1R     08.01.54     1.749
  LATCHWAITPID=         0 CMD=BPXVCLNY
TCPIPOE  TCPIPMVS 0053         29          1 MR     08.02.22  19950.289
  LATCHWAITPID=         0 CMD=EZBTTMST
STC      RMFGAT   0059   33554462          1 1R     08.03.20  60749.552
  LATCHWAITPID=         0 CMD=ERB3GMFC
STC      EJWSBKR  0058         31          1 1FI    08.03.18     1.796
  LATCHWAITPID=         0 CMD=EJWSCOM
OMVSKERN INETD1   001D         32          1 1FI    08.03.18     1.512
  LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
STC      PMAPOE1  0022         34          1 1FI    08.03.28     1.657
  LATCHWAITPID=         0 CMD=OPORTMAP
STC      PORTMAP  005B         35          1 1FI    08.03.28     1.832
  LATCHWAITPID=         0 CMD=PORTMAP
STC      RXPROC   005C         36          1 1FI    08.03.28     1.813
  LATCHWAITPID=         0 CMD=RSHD
FTPDOE   FTPDOE1  001E         39          1 1FI    08.03.29     1.684
  LATCHWAITPID=         0 CMD=FTPD
NICHOLS  NICHOLS  01F7   16777394          1 1RI    14.17.11     1.443
  LATCHWAITPID=         0 CMD=EXEC
```

Figure 12-5 Command to display information about the active processes

To display information about all users of z/OS UNIX, use the following command:

```
D OMVS,ASID=ALL
```

This will show all z/OS UNIX processes with MVS job name and address space ID (ASID). Information about a specific address space can be displayed by:

```
D OMVS,ASID=31
```

The first line of the display shows the status of z/OS UNIX (Active) and the name of the BPXPRMxx parmlib member in use (BPXPRM00). The following status can be displayed:

ACTIVE	OMVS is currently active.
NOT STARTED	OMVS was not started.
INITIALIZING	OMVS is initializing.
TERMINATING	OMVS is terminating.
TERMINATED	OMVS has terminated.
ETC/INIT WAIT	OMVS is waiting for the /etc/init or /usr/sbin/init program to complete initialization.

The remaining information shown about each z/OS UNIX user is:

USER	The user ID of the process.
JOBNAME	The job name of the process.
ASID	The address space ID for the process address space.

PID	The process ID, in decimal, of the process.
PPID	The parent process ID, in decimal, of the process.
STATE	The state of the process or of the most recently created thread in the process. The codes on the figure have the following meanings:

1	Process state is for a single thread process.
F	File system kernel wait.
I	Swapped out.
K	Other kernel wait (for example, pause or sigsuspend).
M	Process state is for multiple threads and pthread_create was not used to create multiple threads. Process state is obtained from the most recently created thread.
R	Running (not kernel wait).

For all other state codes, see *MVS/ESA System Messages, Volume 2* for information about the BPXO001I message.

START	The time, in hours, minutes, and seconds, when the process was started.
CT_SECS	The total execution time for the process in seconds in the format ssssss.hhh. After approximately 11.5 days, this field will overflow. When an overflow occurs, the field is displayed as ******.***
LATCHWAITPID=	Either zero or the latch process ID, in decimal, for which this process is waiting.
CMD=	The command that created the process. Truncated to 40 characters and converted to uppercase.
SERVER=	The name of the server process.
AF=	The number of active server file tokens.
MF=	The maximum number of active server file tokens allowed.
TYPE=	One of the following:

FILE	A network file server
LOCK	A network lock server

SERVERLOCKS=	The number of active lock processes for this server.
SERVERMAXLOCKS=	The maximum number of active lock processes for this server allowed.

12.6 Display process information

```
D OMVS,PID=16777394
BPXO040I 16.32.57 DISPLAY OMVS 753
OMVS      000F ACTIVE          OMVS=(8A)
USER      JOBNAME  ASID        PID       PPID STATE   START      CT_SECS
NICHOLS   NICHOLS  01F7    16777394         1 1RI    14.17.11      1.443
  LATCHWAITPID=         0 CMD=EXEC
 THREAD_ID         TCB@     PRI_JOB  USERNAME   ACC_TIME SC  STATE
 0BA6C1A000000000 007F3338                          .022 NOP  Y
 0BA6D96000000001 007E3460                          .138 LST  Y
```

Figure 12-6 Command to display a specific process

The operator can display the status of a particular OMVS process and its threads with command:

```
D OMVS,PID=xxxxxx
```

This will show information about each thread of a multi-threaded application. The thread ID would be needed to cancel an individual thread.

The first line of the display shows the status of z/OS UNIX (Active) and the name of the BPXPRMxx parmlib member in use (BPXPRM00). The following information about the threads is displayed:

THREAD_ID	The thread ID, in hexadecimal, of the thread.
TCB@	The address of the TCB that represents the thread.
PRI_JOB	The job name from the current primary address space if different from the home address space, otherwise blank. This is only accurate if the thread is in a wait, otherwise it is from the last time that the status was saved.
USERNAME	The username of the thread if a task level security environment created by pthread_security_np exists, otherwise blank.
ACC_TIME	The accumulated TCB time in seconds in the format ssssss.hh. When this value exceeds 11.5 days of execution time this field will overflow. When an overflow occurs, the field is displayed as ******.***.
SC	The current or last syscall request.
STATE	The state of the thread as follows.

A	Message queue receive wait
B	Message queue send wait
C	Communication system kernel wait
D	Semaphore operation wait
E	Quiesce frozen
F	File system kernel wait
G	MVS Pause wait
K	Other kernel wait (for example, pause or sigsuspend)
J	The thread was pthread created rather than dubbed
N	The thread is medium weight
O	The thread is asynchronous and medium weight
P	Ptrace kernel wait

Q	Quiesce termination wait
R	Running (not kernel wait)
S	Sleeping
U	Initial process thread (heavy weight and synchronous)
V	Thread is detached
W	Waiting for child (wait or waitpid callable service)
X	Creating new process (fork callable service is running)
Y	Thread is in an MVS wait

12.7 Display the kernel address space

```
D A,OMVS
IEE115I 10.21.16 2000.157 ACTIVITY 973
 JOBS     M/S     TS USERS    SYSAS    INITS   ACTIVE/MAX VTAM     OAS
00011    00034    00001      00030    00038    00001/00050       00023
 OMVS     OMVS     OMVS     NSW  *   A=000F   PER=NO   SMC=000
                                     PGN=N/A  DMN=N/A  AFF=NONE
                                     CT=023.135S  ET=00122.24
                                     WKL=SYSTEM   SCL=SYSSTC   P=1
                                     RGP=N/A      SRVR=NO  QSC=NO
                                     ADDR SPACE ASTE=3132C3C0
                                     DSPNAME=BPXDQ001 ASTE=26540F00
                                     DSPNAME=SYSZBPXV ASTE=0A1E5800
                                     DSPNAME=SYSZBPXQ ASTE=0A1E5780
                                     DSPNAME=SYSZBPXC ASTE=0A1E5700
                                     DSPNAME=SYSZBPXU ASTE=0A1E5680
                                     DSPNAME=HFSDSP04 ASTE=26540680
                                     DSPNAME=HFSDSP03 ASTE=26540600
                                     DSPNAME=HFSDSP02 ASTE=26540580
                                     DSPNAME=HFSDSP01 ASTE=26540500
                                     DSPNAME=SYSZBPX3 ASTE=26540100
                                     DSPNAME=SYSIGWB1 ASTE=26540280
                                     DSPNAME=BPXSMBIT ASTE=1023E900
                                     DSPNAME=SYSZBPX2 ASTE=0A1E5480
                                     DSPNAME=SYSZBPX1 ASTE=31337380
```

Figure 12-7 Command to display the OMVS (kernel) address space

The operator can display the kernel address space and its associated dataspaces with the command:

```
D A,OMVS
```

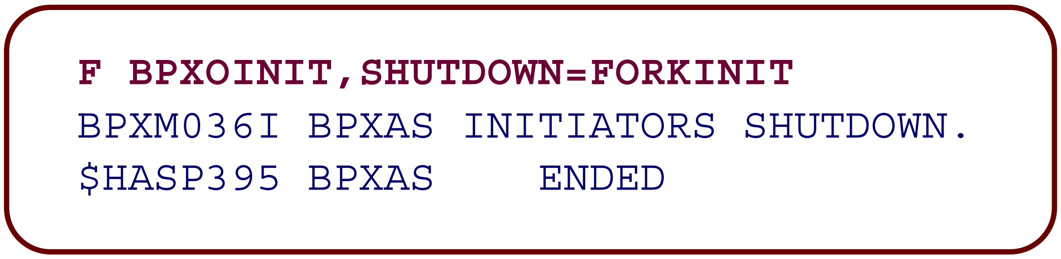
This display shows the OMVS kernel address space identifier (ASID) and the dataspace names associated with the kernel. The system uses these dataspaces as follows:

SYSZBPX1	For kernel data (including CTRACE buffers). The CTRACE buffers are automatically included in the dump and need not be explicitly added to a **DUMP** command or a SLIP trap.
SYSZBPX2	For file system data
SYSZBPX3	For pipes
SYSIGWB1	For byte-range locking
SYSGFU01	For file system adapter
SYSZBPXU	For AF_UNIX sockets
SYSZBPXC	For common INET sockets
SYSZBPXL	For local AF_INET sockets
HFSDPS01	For DFSMS 1.5 4KB buffers pools (and index data)
HFSDPS02	For DFSMS 1.5 16KB buffers pools
HFSDPS03	For DFSMS 1.5 64KB buffers pools
HFSDPS04	For DFSMS 1.5 256KB buffers pools

These dataspaces will need to be included in a dump if they have information that could be useful in analyzing a problem.

The kernel dataspace, SYSZBPX1, is always needed. Dump other dataspaces if there is reason to believe they contain data that could be useful in analyzing a problem.

12.8 Stopping BPXAS address spaces



```
F BPXOINIT,SHUTDOWN=FORKINIT
BPXM036I BPXAS INITIATORS SHUTDOWN.
$HASP395 BPXAS    ENDED
```

Figure 12-8 Command to shut down all BPXAS address spaces

When closing down a system prior to IPL, the **`F BPXOINIT,SHUTDOWN=FORKINIT`** command should be issued prior to stopping JES2. This is because the WLM BPXAS address spaces remain active for 30 minutes after use, and could therefore delay JES2 shutdown for an unacceptable period. The **`F BPXOINIT,SHUTDOWN=FORKINIT`** command terminates inactive BPXAS address spaces prematurely.

Attempting to shut down active BPXAS address spaces will result in a the following message:

```
BPXM037I BPXAS INITIATOR SHUTDOWN DELAYED
```

12.9 BPXSTOP (from Tools and Toys)

```
S BPXSTOP
$HASP100 BPXSTOP  ON STCINRDR
$HASP373 BPXSTOP  STARTED
BPXF024I (IBMUSER) Attempting to terminate active processes with 942
SIGTERM
BPXF024I (IBMUSER) Unmounting all file systems  943
IEF196I IGD104I NEILOC.TC5.HFS                                  RETAINED,
IEF196I DDNAME=SYS00003
IEF196I IGD104I OMVS.TC5.ETC                                    RETAINED,
IEF196I DDNAME=SYS00002
IEF196I IGD104I OMVS.TC5.TRNRS1.ROOT.HFS
RETAINED,
IEF196I DDNAME=SYS00001
$HASP395 BPXSTOP  ENDED
```

Figure 12-9 The BPXSTOP program from the z/OS UNIX Web site

The BPXSTOP tool is available from the z/OS UNIX System Services Tools and Toys Web page at:

http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html

Its purpose is to close down all active UNIX processes, then UNMOUNT all the file systems including the root. It is designed to be used prior to an IPL (just before an **F BPXOINIT,SHUTDOWN=FORKINIT** is issued).

Some reasons you may choose to use it include:

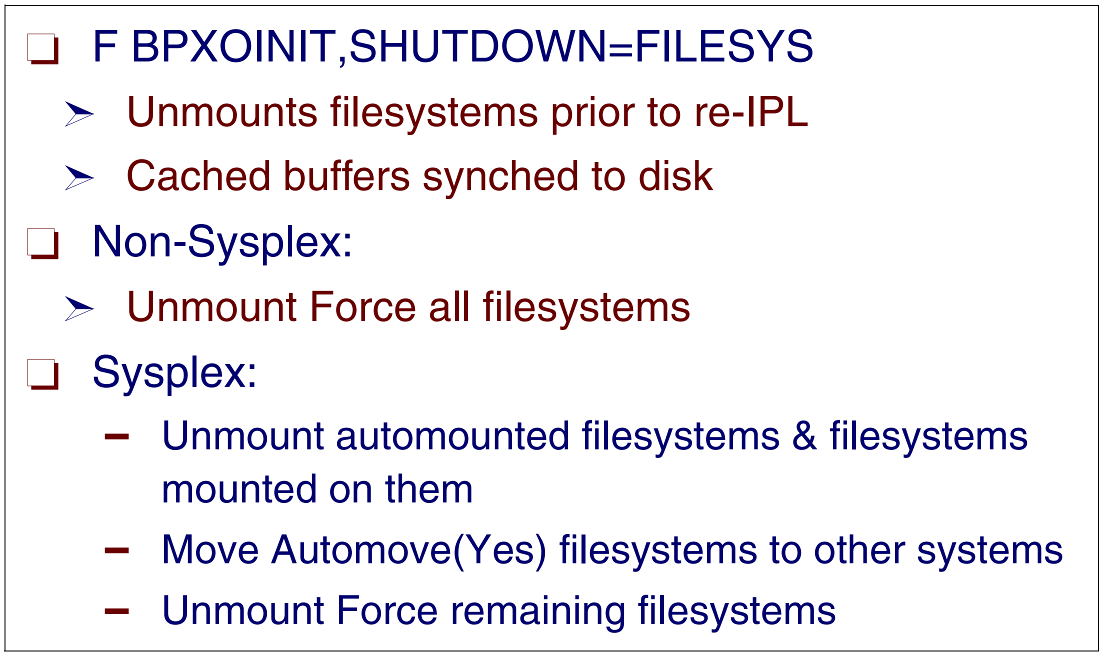
- To ensure clean shutdown of all tasks prior to IPL.
- To free up all the file systems to perform some pervasive storage management tasks.

Visit the referenced Web site to learn more about implementing the BPXSTOP solution.

In z/OS V1R2, a new keyword was added for SHUTDOWN= FILESYS. This new keyword specifies that file systems are to be unmounted. For Shared HFS, the file systems that are OWNED by the system where the command was issued will be:

1. Unmounted if the file system was automounted or mounted on an automounted file system.
2. Moved to another system if it is an automove(yes) file system.
3. Unmounted when (1) or (2) do not apply.

12.10 LFS soft shutdown



- ❑ F BPXOINIT,SHUTDOWN=FILESYS
  - ➢ Unmounts filesystems prior to re-IPL
  - ➢ Cached buffers synched to disk
- ❑ Non-Sysplex:
  - ➢ Unmount Force all filesystems
- ❑ Sysplex:
  - – Unmount automounted filesystems & filesystems mounted on them
  - – Move Automove(Yes) filesystems to other systems
  - – Unmount Force remaining filesystems

Figure 12-10 A command to unmount all file systems

A function to do a soft shutdown for mounted file systems is provided through the MVS operator console Modify command in z/OS V1R2:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

This function is retrofitted back to OS/390 V2R7 with APAR OW48199.

Currently, it is available on the z/OS UNIX System Services home page:

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

This function is also on the Tools and Toys page: under the heading z/OS UNIX tools you can find a tool called BPXSTOP. This tool is used to kill processes and unmount file systems after normal shutdown procedures and termination commands for UNIX products have been issued. Not all servers and applications respond well to terminating signals. You should make every attempt to shut down all UNIX work you can, including and especially TCP/IP and NFS client and server, prior to using this function. Using BPXSTOP may not harden the data to disk when the file systems are unmounted.

Now there is a new and supported way to help terminate UNIX activity on the system. In z/OS V1R2 there is an enhancement to the existing **`F BPXOINIT,SHUTDOWN=`** command, which provides a method for unmounting all file systems and hardening the cached buffers to disk.

Many of the different products running under z/OS UNIX System Services, like NFS for z/OS and Communication Server, have their own shutdown procedures. These procedures should be used first to shut down the different products, before any z/OS UNIX System Services termination commands are used.

The distinction between the shared HFS file systems in a Sysplex and non-Sysplex file systems is based on whether SYSPLEX(YES) is specified in the BPXPRMxx parmlib member, and the systems are participating in a shared HFS.

If it is a non-sysplex environment, all file systems will be unmounted with the FORCE operand.

In a sysplex environment, all automounted file systems and file systems mounted on the system you shut down will be unmounted. For file systems mounted with AUTOMOVE parameter set, the ownership is moved to another system in the sysplex. The remaining file systems will be unmounted FORCE.

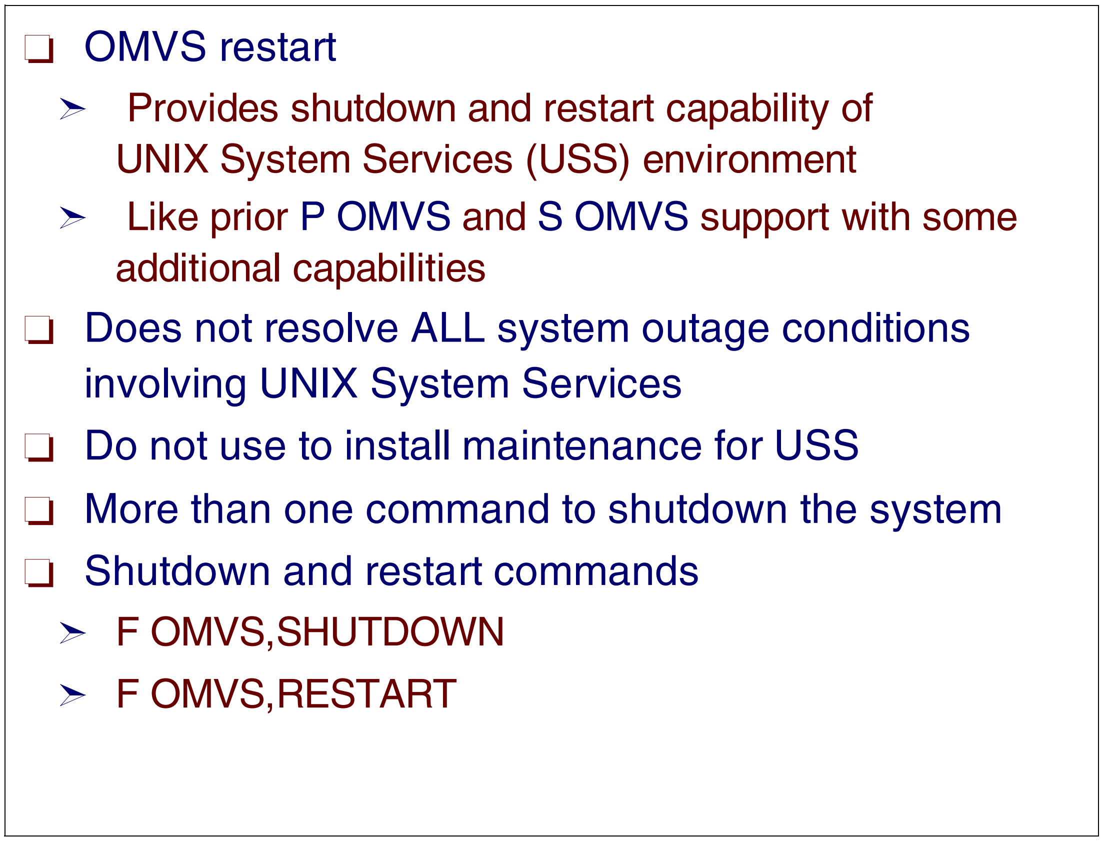
F BPXOINIT command

The Modify MVS console command **`F BPXOINIT SHUTDOWN`** is now enhanced with the FILESYS parameter:

```
F BPXOINIT SHUTDOWN=FILESYS
```

where the operand `SHUTDOWN=FILESYS` unmounts the UNIX System Services file systems as described.

12.11 z/OS UNIX shutdown (z/OS V1R3)



- OMVS restart
  - Provides shutdown and restart capability of UNIX System Services (USS) environment
  - Like prior P OMVS and S OMVS support with some additional capabilities
- Does not resolve ALL system outage conditions involving UNIX System Services
- Do not use to install maintenance for USS
- More than one command to shutdown the system
- Shutdown and restart commands
  - F OMVS,SHUTDOWN
  - F OMVS,RESTART

Figure 12-11 Commands to shutdown and restart z/OS UNIX without re-IPL

z/OS V1R3 has introduced the possibility to shut down and re-initialize the z/OS UNIX environment without the need to IPL the system. This new OMVS option will shut down z/OS UNIX and all the processes that are running under it.

OMVS shutdown allows you to do some reconfiguration that would otherwise have required an IPL, for example:

- If you do a reconfiguration of a system to go from a non-shared HFS system to a shared HFS system
- When you implement a new file structure

There are some restrictions and limitations that are not allowed with the OMVS shutdown. In these cases, you will need to IPL the system as usual. These limitations include:

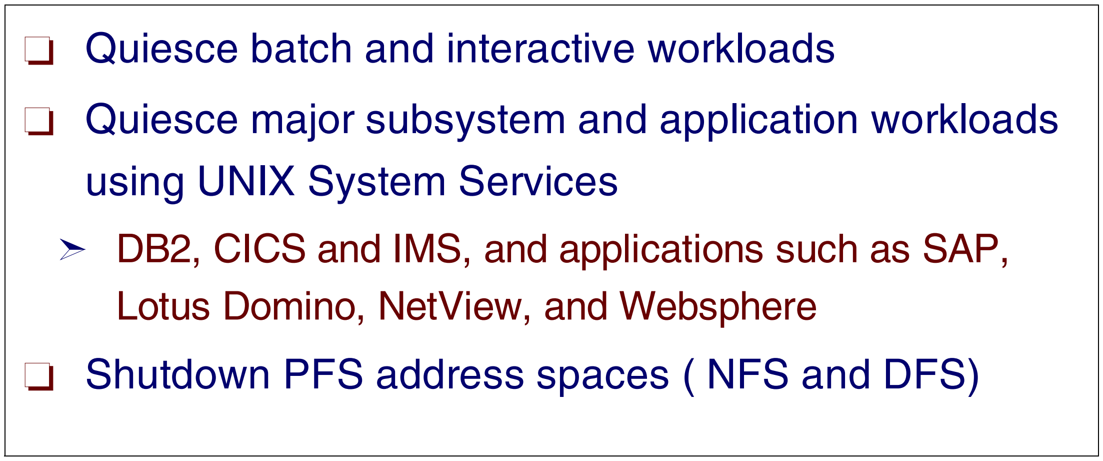
- During the cleanup of the resources for z/OS UNIX as part of the shutdown process, some internal failures cannot be resolved using this support due to their severity.
- OMVS shutdown support cannot be used to install maintenance against z/OS UNIX because some of the modules are maintained across the shutdown and restart process.
- Installations should avoid using the OMVS restart support as a way to shutdown the system with one single command. This will cause some unexpected abnormal terminations of address spaces using UNIX System Services that are not shut down in the manner they recommend.

- OMVS restart support is not intended to be used in an unlimited manner to shut down and restart, because some system resources can be lost during the shutdown phase and due to the disruption it causes to the system.

In order to support OMVS shutdowns, there is a new **`Modify`** command; support for the OMVS address space has been introduced to provide the ability to shut down and then restart the z/OS UNIX environment with the following commands:

```
F OMVS,SHUTDOWN
F OMVS,RESTART
```

12.12 Recommended shutdown procedures



- ❑ Quiesce batch and interactive workloads
- ❑ Quiesce major subsystem and application workloads using UNIX System Services
  - ➢ DB2, CICS and IMS, and applications such as SAP, Lotus Domino, NetView, and Websphere
- ❑ Shutdown PFS address spaces (NFS and DFS)

Figure 12-12 Recommended procedures to shut down OMVS

Quiesce your batch and TSO workloads. Having batch jobs and TSO users running during the shutdown may cause these jobs to experience unexpected signals or abends. Additionally, these jobs and users may end up being hung, waiting for z/OS UNIX services to be restarted, if they first access z/OS UNIX services during a shutdown.

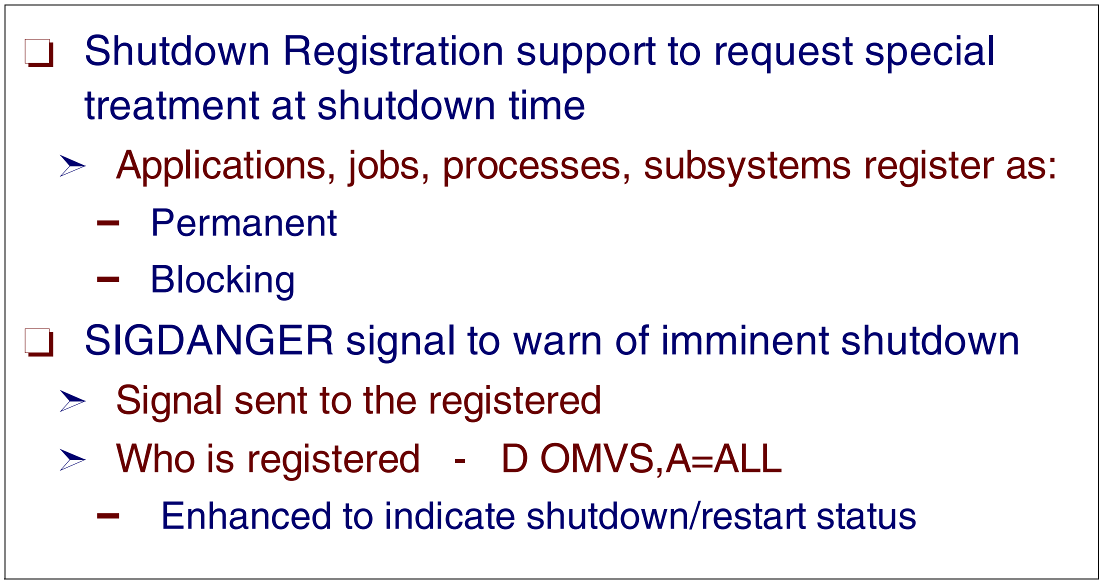
Quiesce those application and subsystem workloads using z/OS UNIX services in the manner that each application or subsystem recommends. Doing so will allow subsystems such as DB2, CICS, and IMS™, and applications like SAP, Lotus® Domino, NetView, and WebSphere to be quiesced in a more controlled manner than this facility will provide.

Unmount all remotely mounted file systems such as those managed by NFS. Doing so will prevent these file systems from losing data.

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 may terminate abnormally when the shutdown takes place. Existing colony PFS address spaces will be shut down. This may include NFS, for example.

Note: You can use the **`D OMVS,A=ALL`** operator command to determine which applications, if any, require quiescing.

12.13 Application registration



- Shutdown Registration support to request special treatment at shutdown time
  - Applications, jobs, processes, subsystems register as:
    - Permanent
    - Blocking
- SIGDANGER signal to warn of imminent shutdown
  - Signal sent to the registered
  - Who is registered - D OMVS,A=ALL
    - Enhanced to indicate shutdown/restart status

Figure 12-13 Processes register for shutdown processing

New registration support has been introduced to allow an application to request special treatment when a shutdown is initiated, and to request to receive a new SIGDANGER signal as a warning that shutdown has been initiated and is imminent.

The registration support allows requesting of special treatment in case of shutdown. Different kinds of registrations can be implemented, as follows:

- A process or job registered as “permanent” is not taken down across the shutdown and restart process. Its process-related resources are checkpointed at shutdown time and reestablished at restart time, so the registered permanent process or job can survive the shutdown.
- A process or job registered as “blocking” delays shutdown until it de-registers or ends. This gives the ability for an application to quiesce itself in a more controlled manner before UNIX System Service starts taking down all processes.
- A process or job registered for “notification” is notified that the shutdown process is being planned via a SIGDANGER signal.

The following command has been modified to include information about what type of registration a specific process has:

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

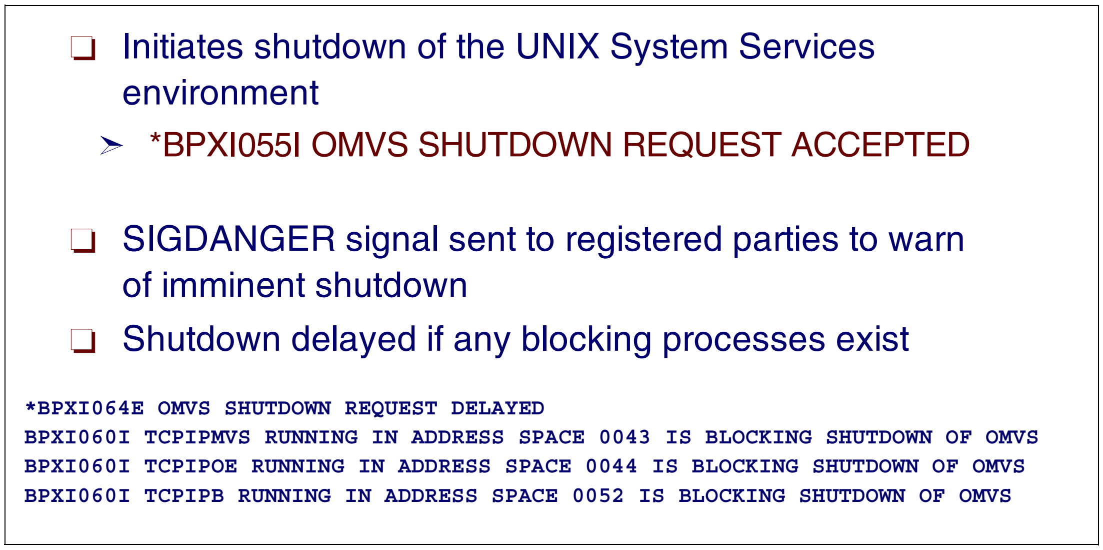
12.14 Display application registration

```
D OMVS,A=ALL
BPXO040I 10.02.18 DISPLAY OMVS 543
OMVS     000F ACTIVE          OMVS=(3A)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 003C          1          0 MRI--- 07.58.59       .18
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                     AF=     0 MF=00000 TYPE=FILE
STC      MVSNFSC5 003B   16908290          1 1R---- 07.59.13       .06
  LATCHWAITPID=         0 CMD=GFSCMAIN
STC      MVSNFSC5 003B   50462724          1 1R---- 07.59.12       .06
  LATCHWAITPID=         0 CMD=BPXVCLNY
STC      MVSNFSC5 003B   50462728          1 1A---- 07.59.14       .06
  LATCHWAITPID=         0 CMD=BPXVCMT
OMVSKERN SYSLOGD5 0041     131081          1 1FI--- 07.59.06       .13
  LATCHWAITPID=         0 CMD=/usr/sbin/syslogd -f /etc/syslog.conf
STC      RMFGAT   0046   84017164          1 1R---P 08.00.01     83.91
  LATCHWAITPID=         0 CMD=ERB3GMFC
TCPIPMVS TCPIPMVS 0043     131085          1 MR---B 08.00.06      8.35
  LATCHWAITPID=         0 CMD=EZBTCPIP
TCPIPMVS TCPIPMVS 0043     131086          1 1R---B 08.00.12      8.35
  LATCHWAITPID=         0 CMD=EZBTTSSL
TCPIPMVS TCPIPMVS 0043     131087          1 1R---B 08.00.12      8.35
```

Figure 12-14 Command to display registered processes

As shown in the figure, a character `P` or `B`, indicating permanent or blocked, has been included on the STATE field.

12.15 F OMVS,SHUTDOWN



- ❑ Initiates shutdown of the UNIX System Services environment
  - ➢ *BPXI055I OMVS SHUTDOWN REQUEST ACCEPTED
- ❑ SIGDANGER signal sent to registered parties to warn of imminent shutdown
- ❑ Shutdown delayed if any blocking processes exist

```
*BPXI064E OMVS SHUTDOWN REQUEST DELAYED
BPXI060I TCPIPMVS RUNNING IN ADDRESS SPACE 0043 IS BLOCKING SHUTDOWN OF OMVS
BPXI060I TCPIPOE RUNNING IN ADDRESS SPACE 0044 IS BLOCKING SHUTDOWN OF OMVS
BPXI060I TCPIPB RUNNING IN ADDRESS SPACE 0052 IS BLOCKING SHUTDOWN OF OMVS
```

Figure 12-15 Issuing the shutdown command

The shutdown starts by issuing the **`F OMVS,SHUTDOWN`** command and then the following steps are done:

1. Once the shutdown command has been accepted, a `BPXI055I` is issued:

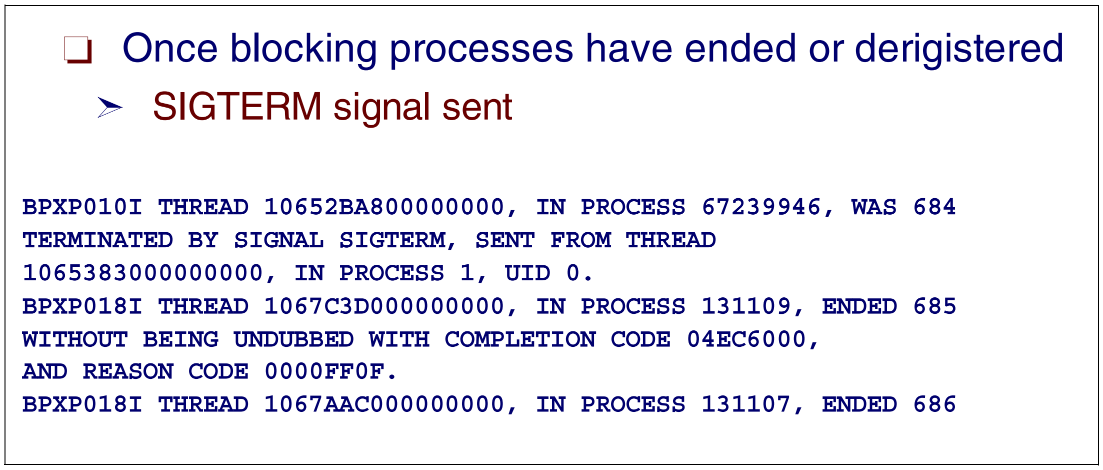
   ```
   *BPXI055I OMVS SHUTDOWN REQUEST ACCEPTED
   ```

   SIGDANGER signals are sent to all processes registered for receiving SIGDANGER signal.

If any blocking processes are found, shutdown is delayed until these processes end or de-register as blocking, or if a **`F OMVS,RESTART`** command is done to restart. If these blocking processes do not end or deregister in a reasonable amount of time, message `BPXI064E` is displayed to the console, indicating shutdown is delayed. In our tests, twelve seconds after the shutdown command has been accepted, the `BPXI064E` message was issued. Message `BPXI060I` was also issued for each process found to be holding up the shutdown. This message identified the job and address space involved.

Attention: Use the **`F OMVS,SHUTDOWN`** command carefully because this method will take down other system address spaces. As a result, some system-wide resources may 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 re-IPL.)

12.16 Blocking processes completion



- ❑ Once blocking processes have ended or derigistered
  - ➢ SIGTERM signal sent

```
BPXP010I THREAD 10652BA800000000, IN PROCESS 67239946, WAS 684
TERMINATED BY SIGNAL SIGTERM, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
BPXP018I THREAD 1067C3D000000000, IN PROCESS 131109, ENDED 685
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 04EC6000,
AND REASON CODE 0000FF0F.
BPXP018I THREAD 1067AAC000000000, IN PROCESS 131107, ENDED 686
```

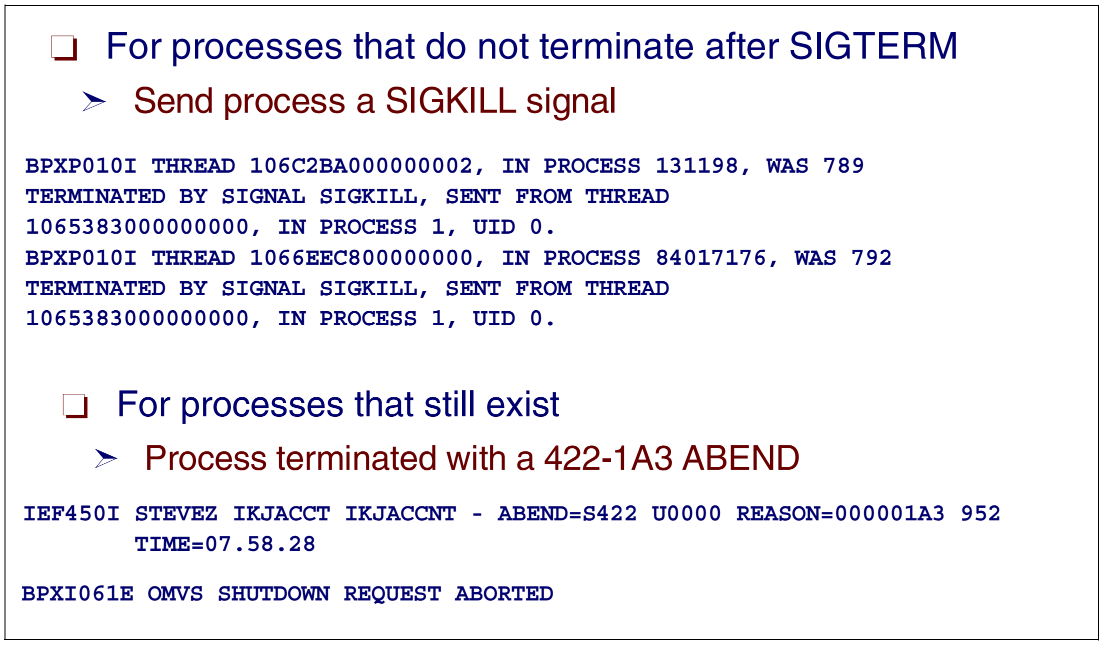
Figure 12-16 Messages when blocking processes complete

Some applications may 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 may 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 terminate 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 terminated.

Once all blocking processes have ended or deregistered as blocking, the shutdown follows by sending a SIGTERM signal to each non-permanent process found and the following messages are received, as shown in the figure.

12.17 Shutdown processing completion



- For processes that do not terminate after SIGTERM
  - Send process a SIGKILL signal

```
BPXP010I THREAD 106C2BA000000002, IN PROCESS 131198, WAS 789
TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
BPXP010I THREAD 1066EEC800000000, IN PROCESS 84017176, WAS 792
TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
```

- For processes that still exist
  - Process terminated with a 422-1A3 ABEND

```
IEF450I STEVEZ IKJACCT IKJACCNT - ABEND=S422 U0000 REASON=000001A3 952
        TIME=07.58.28

BPXI061E OMVS SHUTDOWN REQUEST ABORTED
```

Figure 12-17 Steps to terminate active processes

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

- Start by sending a SIGTERM signal:

  ```
  kill -s TERM pid
  ```

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

  ```
  kill -s KILL pid
  ```

  where pid is the process identifier.

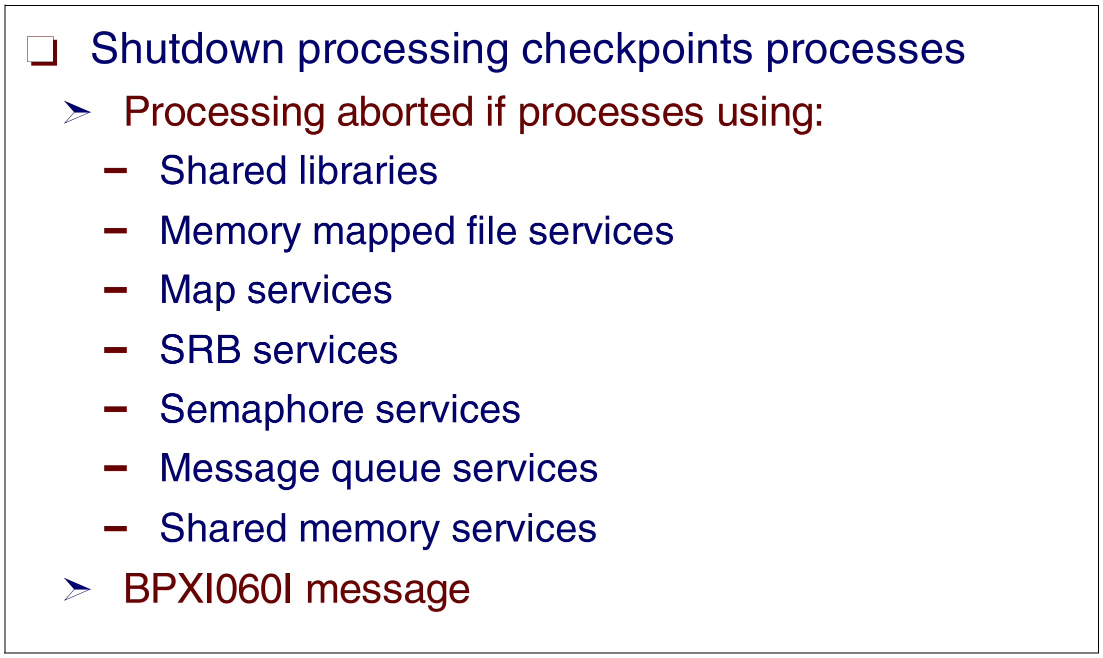
If some of the processes still exist after both of these signals are sent, they are terminated with a 422-1A3 ABEND.

```
IEF450I STEVEZ IKJACCT IKJACCNT - ABEND=S422 U0000 REASON=000001A3 952
        TIME=07.58.28
```

If after all of these steps, some non-permanent processes still exist, the shutdown request is aborted and a `BPXI061E` message is issued:

```
BPXI061E OMVS SHUTDOWN REQUEST ABORTED
```

12.18 Permanent processes



- Shutdown processing checkpoints processes
  - Processing aborted if processes using:
    - Shared libraries
    - Memory mapped file services
    - Map services
    - SRB services
    - Semaphore services
    - Message queue services
    - Shared memory services
  - BPXI060I message

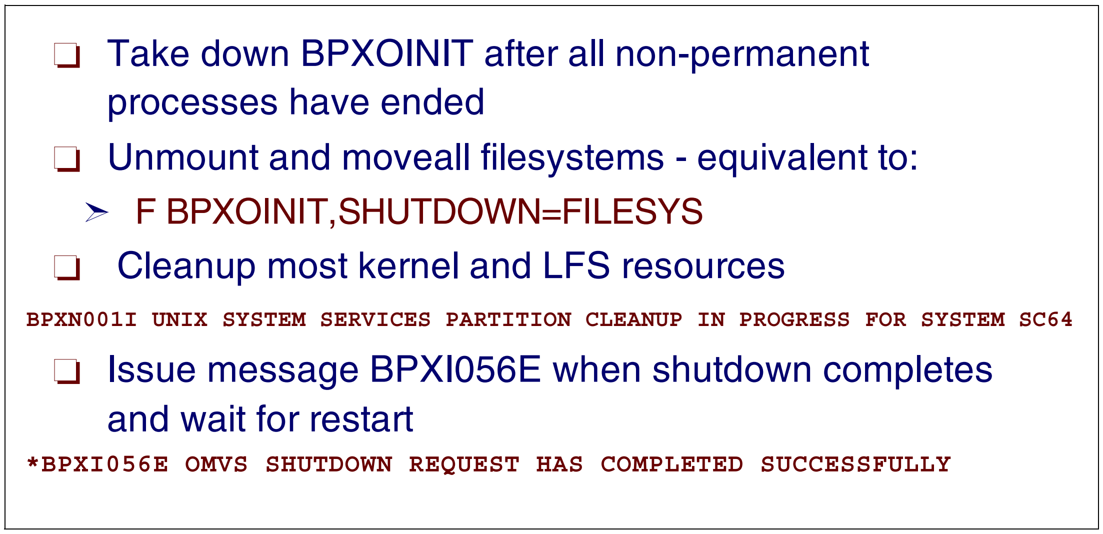
Figure 12-18 Permanent processes termination

After non-permanent processes have been taken down, the shutdown process continues trying to checkpoint all the permanent processes. A permanent process cannot be checkpointed, however, if it is using any of the following resources:

- Shared libraries
- Memory mapped file services
- Map services
- SRB services
- Semaphore services
- Message queue services
- Shared memory services

A permanent process found using any of these resources will cause shutdown to be aborted and message `BPXI060I` is issued indicating what resource for which job is causing the problem.

12.19 Shutdown processing final cleanup



- ❑ Take down BPXOINIT after all non-permanent processes have ended
- ❑ Unmount and moveall filesystems - equivalent to:
  - ➢ F BPXOINIT,SHUTDOWN=FILESYS
- ❑ Cleanup most kernel and LFS resources

```
BPXN001I UNIX SYSTEM SERVICES PARTITION CLEANUP IN PROGRESS FOR SYSTEM SC64
```

- ❑ Issue message BPXI056E when shutdown completes and wait for restart

```
*BPXI056E OMVS SHUTDOWN REQUEST HAS COMPLETED SUCCESSFULLY
```

Figure 12-19 Final cleanup of shutdown processing

After all non-permanent processes have ended, BPXOINIT is taken down with a 422-1A3 abend.

File systems on the system where the shutdown was issued are immediately unmounted; data is synched to disk as a result.

For shared HFS, 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(YES) 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.

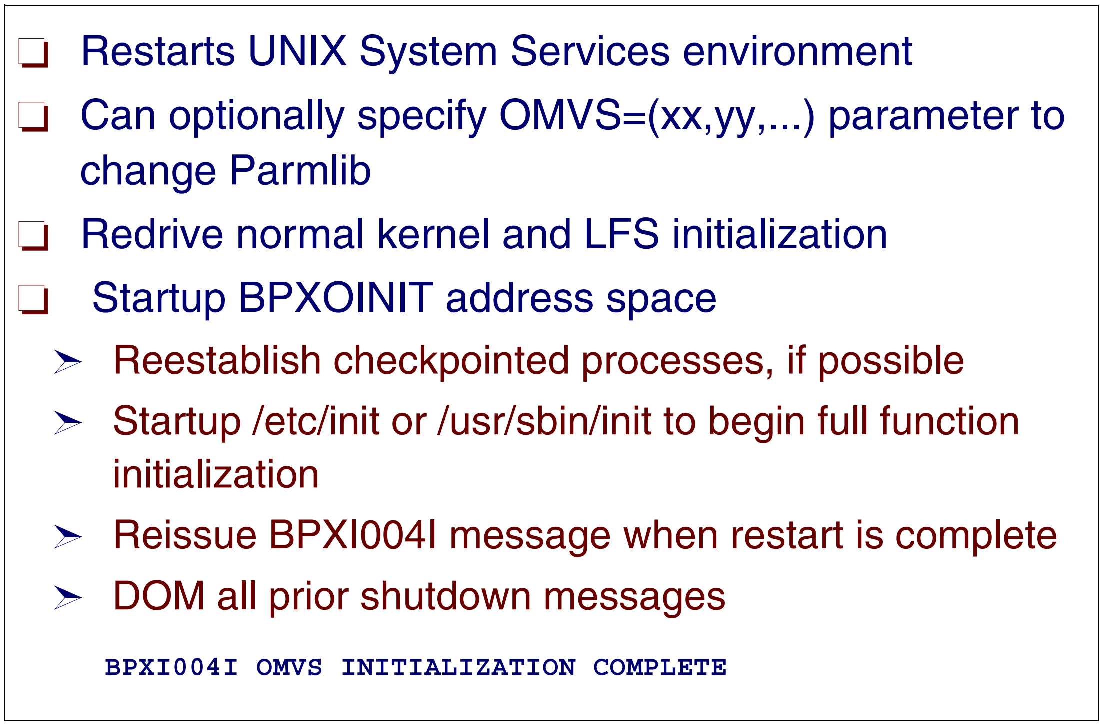
The shutdown should be done prior to an IPL. It replaces BPXSTOP.

On the system that you are preparing to shut down, issue the following command:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

All file systems are unmounted and potentially moved to another system. If for some reason it is not possible to unmount some file systems, a `BPXI066E` message is issued and shutdown will proceed to the next phase.

12.20 F OMVS,RESTART



- ❑ Restarts UNIX System Services environment
- ❑ Can optionally specify OMVS=(xx,yy,...) parameter to change Parmlib
- ❑ Redrive normal kernel and LFS initialization
- ❑ Startup BPXOINIT address space
  - ➢ Reestablish checkpointed processes, if possible
  - ➢ Startup /etc/init or /usr/sbin/init to begin full function initialization
  - ➢ Reissue BPXI004I message when restart is complete
  - ➢ DOM all prior shutdown messages

```
BPXI004I OMVS INITIALIZATION COMPLETE
```

Figure 12-20 Command to restart z/OS UNIX

The **F OMVS,RESTART** command restarts the z/OS UNIX environment. This involves the following:

1. Once the restart command has been accepted, indicated by the following message:

   ```
   *BPXI058I OMVS RESTART REQUEST ACCEPTED
   ```

   the first step in the restart process is to re-initialize the kernel and LFS. This includes starting up all physical file systems.

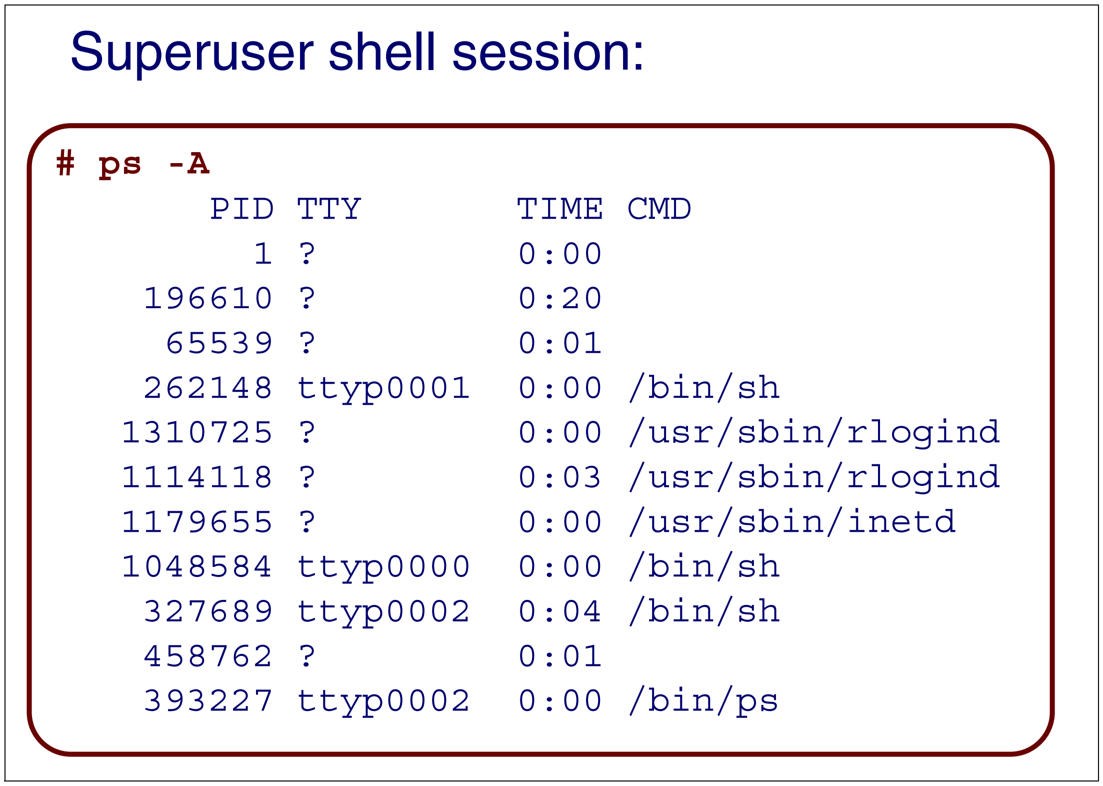
2. BPXOINIT is restarted and it will re-establish itself as process ID 1.
3. BPXOINIT re-establishes the checkpointed processes as follows:

   All checkpointed processes that are still active are re-established and those that are not found are not re-established and will have their checkpointed resources cleaned up.

4. After BPXOINIT completes its initialization, it will restart /etc/init or /usr/sbin/init to begin full function initialization of the z/OS UNIX environment. /etc/init performs its normal startup processing, invoking /etc/rc.
5. After /etc/init has completed full function initialization, a `BPXI0041` message is issued indicating z/OS UNIX initialization is complete.

   ```
   BPXI004I OMVS INITIALIZATION COMPLETE
   ```

12.21 Display information about processes


Superuser shell session:

```
# ps -A
      PID TTY        TIME CMD
        1 ?          0:00
   196610 ?          0:20
    65539 ?          0:01
   262148 ttyp0001   0:00 /bin/sh
  1310725 ?          0:00 /usr/sbin/rlogind
  1114118 ?          0:03 /usr/sbin/rlogind
  1179655 ?          0:00 /usr/sbin/inetd
  1048584 ttyp0000   0:00 /bin/sh
   327689 ttyp0002   0:04 /bin/sh
   458762 ?          0:01
   393227 ttyp0002   0:00 /bin/ps
```

Figure 12-21 Displaying information about active processes

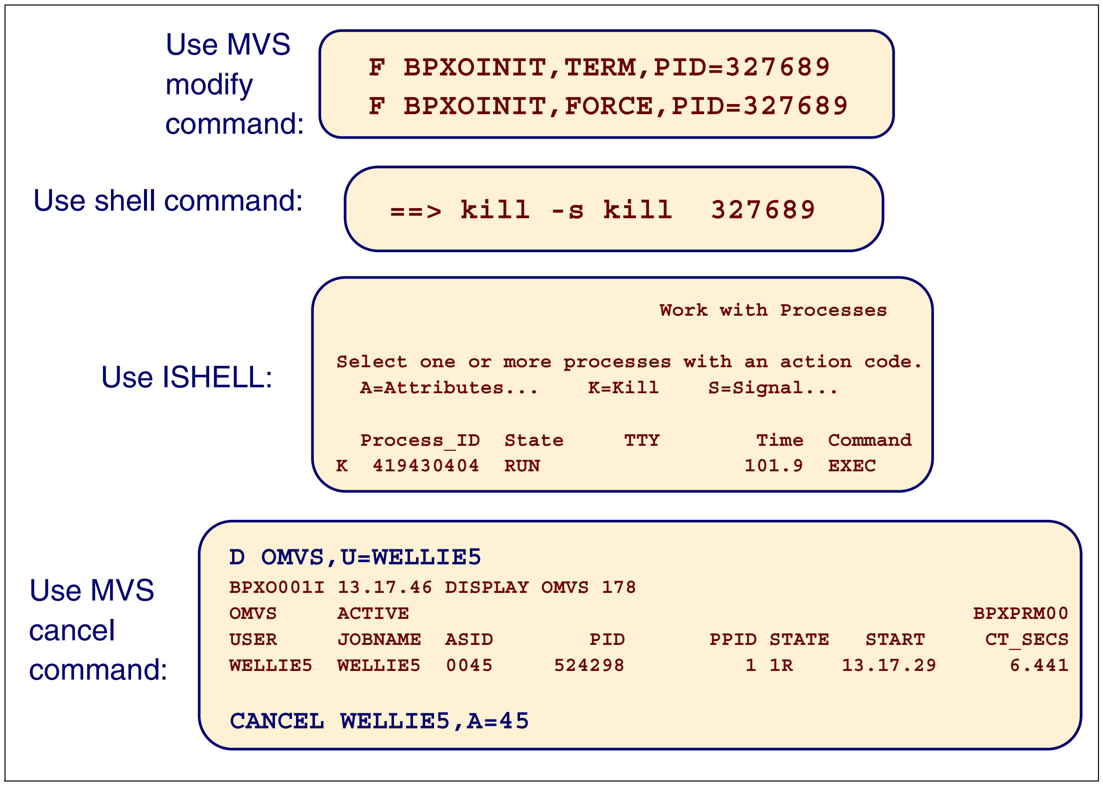
To get information about z/OS UNIX processes, the shell command **ps** can also be used. An z/OS UNIX user can use this command to display information about all active processes.

A superuser can display information about all the processes in the system by using the command **ps -A**.

Figure 12-21 shows an example of a superuser issuing the command **ps -A** to get information about all the active z/OS UNIX processes. This is an alternative to using the **D OMVS** command. If there is a problem with a process, the user or system administrator will first try to stop the process using shell commands. To get the PID of the process, the **ps** command is used.

From the example, you can see that there are three shell sessions because there are three different ttyp000x (1, 2, and 3). There are two process IDs that is using the same pseudo-TTY (ttyp0002). The reason for this is that the **ps** command (/bin/ps) is running in a subshell, but is using the same terminal for output as the shell session (/bin/sh) where the command was issued from. These two PIDs belong to the superuser that issued the **ps -A** command.

12.22 Stop a process



```
F BPXOINIT,TERM,PID=327689
F BPXOINIT,FORCE,PID=327689
```

```
==> kill -s kill  327689
```

```
                              Work with Processes

Select one or more processes with an action code.
  A=Attributes...    K=Kill    S=Signal...

  Process_ID  State     TTY         Time  Command
K  419430404  RUN                  101.9  EXEC
```

```
D OMVS,U=WELLIE5
BPXO001I 13.17.46 DISPLAY OMVS 178
OMVS     ACTIVE                                           BPXPRM00
USER     JOBNAME  ASID        PID        PPID STATE   START     CT_SECS
WELLIE5  WELLIE5  0045     524298           1 1R     13.17.29     6.441

CANCEL WELLIE5,A=45
```

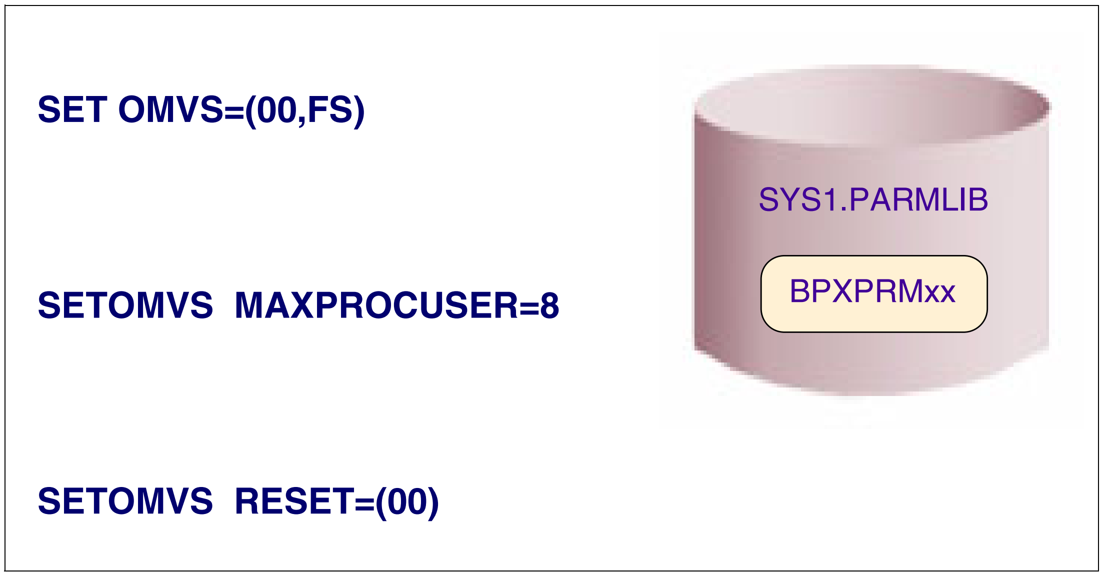
Figure 12-22 Different commands to stop a process

There are four ways to stop a z/OS UNIX process:

1. The operator **MODIFY** command. The TERM option will allow a signal interface routine to receive control before termination. The FORCE option prevents the signal interface routine from receiving control before the process is terminated.
2. The **kill** shell command can be used to cancel a process. A user can cancel his/her own processes, or a system administrator (superuser) has authority to kill other user's processes. TERM sends a SIGTERM signal and KILL sends a SIGKILL signal.

   The **kill** command sends a signal to a process. One of the signals that can be sent is the kill signal, and that is the reason why kill is used twice in the command. The system administrator can use the **ps** command to find the PID of the process. The PID is needed to identify the process in the **kill** command.
3. The ISHELL **k** line command of the Work with Processes menu. You can get to this from the Tools pull-down menu in the ISHELL.
4. The MVS **CANCEL** command can be used to cancel an address space that contains a z/OS UNIX process. If the address space contains multiple processes, **CANCEL** will cause all of them to terminate. **CANCEL** is an operator command.

12.23 Changing OMVS parameter values



```
SET OMVS=(00,FS)

SETOMVS  MAXPROCUSER=8

SETOMVS  RESET=(00)
```

Figure 12-23 Commands to dynamically change BPXPRMxx values

You can change the setting of some of the BPXPRMxx values dynamically using the **`SETOMVS`** or **`SET OMVS`** commands.

The **`SET OMVS`** command lets you dynamically reconfigure the z/OS UNIX system services by specifying one or more BPXPRMxx parmlib members to switch to. You can have multiple parmlib members stored and use them to establish a new configuration. You can only change the values in the list below.

`SETOMVS` lets you change specific values independently of others stored in the same parmlib member.

Changes to system limits take place immediately (for example, MAXPROCSYS). Changes to user limits (for example, MAXTHREADS) are set when a new user enters the system and they last for the length of the user's connection to z/OS UNIX. Values that can be changed are:

MAXPROCSYS - MAXPROCUSER - MAXFILEPROC - MAXFILESIZE - MAXCPUTIME
MAXUIDS - MAXPTYS - MAXRTYS - MAXTHREADTASKS - MAXTHREADS -
MAXMMAPAREA - MAXSHAREPAGES - MAXCORESIZE - MAXASSIZE - All IPC values
FORKCOPY - STEPLIBLIST - USERIDALIASTABLE - PRIORITYPG - PRIORITYGOAL

Make sure you use valid values that can be specified on each of these parameters. There are range limits on some of the variables and some ranges are dynamically calculated based on the parameter's present value.

12.24 Display file system information

```
D OMVS,FILE
BPXO044I 16.37.12 DISPLAY OMVS 760
OMVS     000F ACTIVE          OMVS=(8A)
TYPENAME   DEVICE ----------STATUS----------- MODE QJOBNAME  QPID
TFS             7 ACTIVE                      RDWR
  NAME=/TMP
  PATH=/tmp
  MOUNT PARM=-s 500
HFS            24 ACTIVE                      RDWR
  NAME=OMVS.SC43.SHARKMN
  PATH=/u/sharkmn
HFS            23 ACTIVE                      RDWR
  NAME=VAINI.HFS
  PATH=/vaini
HFS            22 ACTIVE                      RDWR
  NAME=OMVS.SC43.ADSM.LOGS
  PATH=/var/adsm/logs
HFS            21 ACTIVE                      RDWR
  NAME=OMVS.SC43.WAS.JUHA
  PATH=/was/juha
HFS            20 ACTIVE                      RDWR
  NAME=OMVS.SC43.WEB.JUHA
  PATH=/web/juha
HFS            19 ACTIVE                      RDWR
  NAME=OMVS.SC43.JAVA118.V990911.HFS
```

Figure 12-24 Command to display file system information

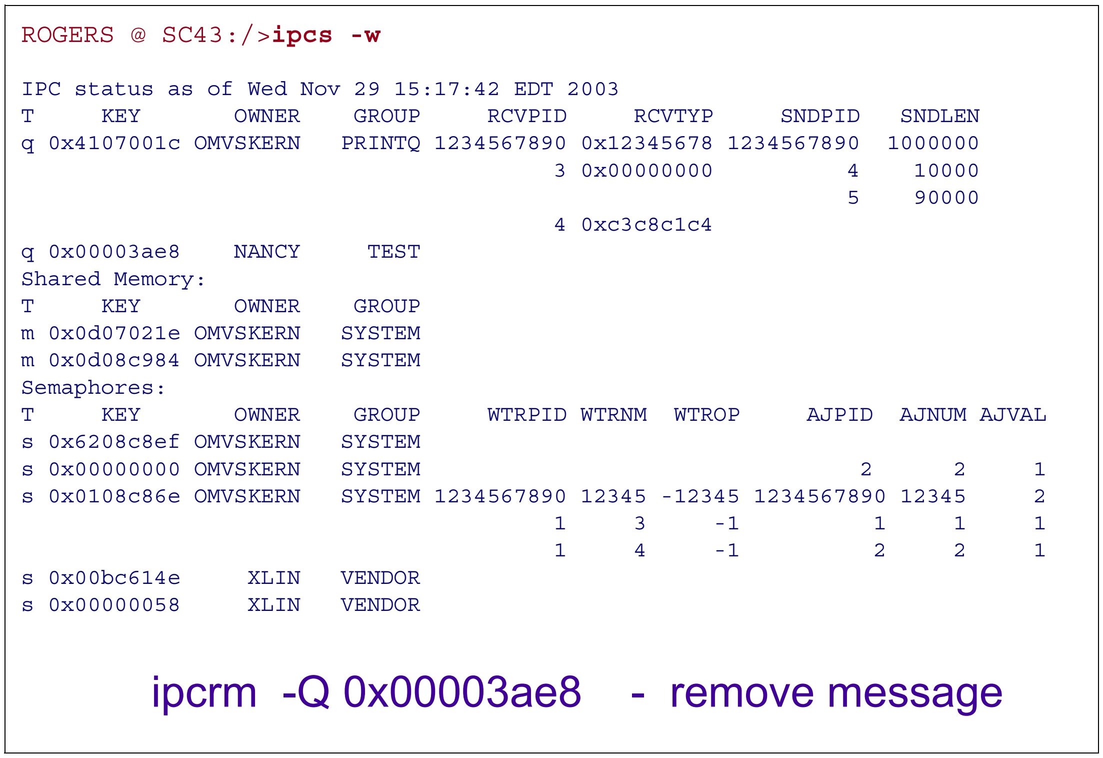
This command provides information about the file systems that are mounted, including the data set name.

Similar information can be gained by using the ISHELL File_Systems menu.

For each file system, the following information will be shown:

TYPENAME	File system type as defined by the FILESYSTYPE statement.
DEVICE	The device value to uniquely identify the device.
STATUS	All file systems are ACTIVE in our example. See the message `BPXO002I` for all the different values status can have.
QJOBNAME	The jobname that quiesced the file system.
QPID	The process ID that quiesced the file system.
NAME=	This is the name of the HFS data set containing the file.
PATH=	The name of the directory where the file system is mounted. The name is truncated to 60 characters. It may be converted to uppercase using the CAPS option.
MOUNT PARM=	The parameter specified to the mount callable service, truncated to 57 characters. It might have been converted to uppercase using the CAPS option.

12.25 Manage interprocess communication



```
ROGERS @ SC43:/>ipcs -w

IPC status as of Wed Nov 29 15:17:42 EDT 2003
T     KEY       OWNER    GROUP     RCVPID     RCVTYP     SNDPID   SNDLEN
q 0x4107001c OMVSKERN   PRINTQ 1234567890 0x12345678 1234567890  1000000
                                        3 0x00000000          4    10000
                                                              5    90000
                                        4 0xc3c8c1c4
q 0x00003ae8    NANCY     TEST
Shared Memory:
T     KEY       OWNER    GROUP
m 0x0d07021e OMVSKERN   SYSTEM
m 0x0d08c984 OMVSKERN   SYSTEM
Semaphores:
T     KEY       OWNER    GROUP     WTRPID WTRNM  WTROP     AJPID  AJNUM AJVAL
s 0x6208c8ef OMVSKERN   SYSTEM
s 0x00000000 OMVSKERN   SYSTEM                                 2      2     1
s 0x0108c86e OMVSKERN   SYSTEM 1234567890 12345 -12345 1234567890 12345     2
                                        1     3     -1          1     1     1
                                        1     4     -1          2     2     1
s 0x00bc614e     XLIN   VENDOR
s 0x00000058     XLIN   VENDOR
```

ipcrm -Q 0x00003ae8   - remove message

Figure 12-25 Command to display interprocess communication information

Figure 12-25 shows an example of the output from an **`IPCS`** command. The **`-w`** option adds information about wait status for message queues and semaphores to the output.

The z/OS UNIX Interprocess Communication (IPC) functions are:

- Shared memory
- Message queues
- Semaphores

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

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

The **`ipcs`** shell command can be used to display the IPC resources in a system, which users own the resources, and which users are waiting for a resource.

You can remove a message queue by using the **`ipcrm`** command. For example, to remove the message queue with `KEY=0x00003ae8` which belongs to user `NANCY` (see the figure), use the command:

```
ipcrm -Q 0x00003ae8
```

12.26 System problems

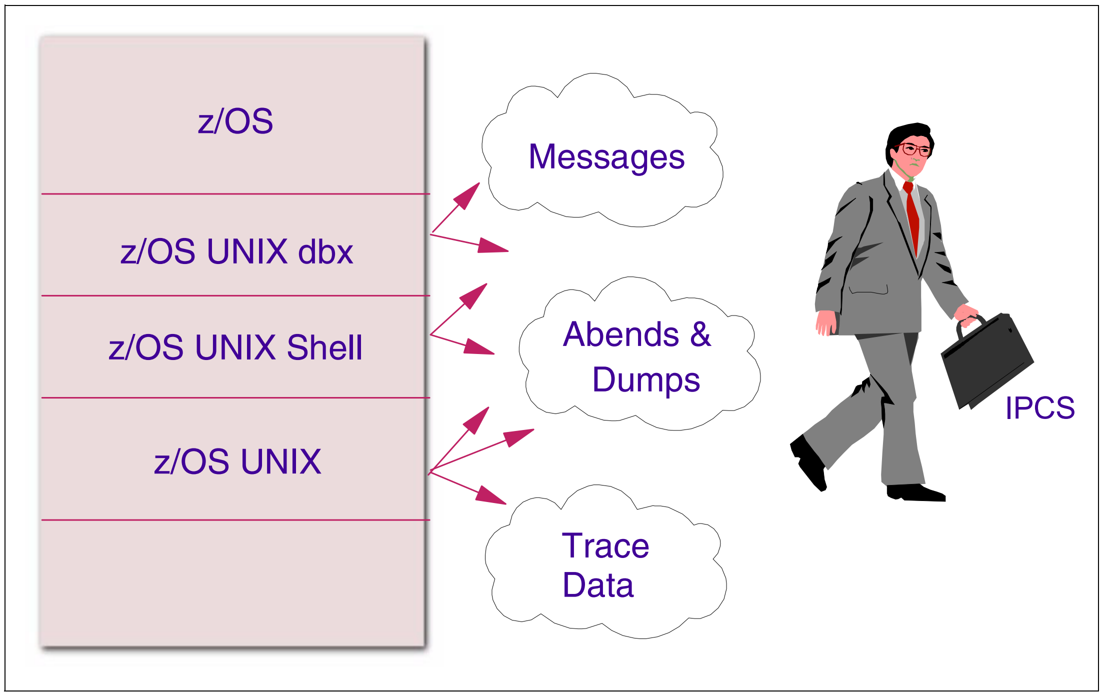


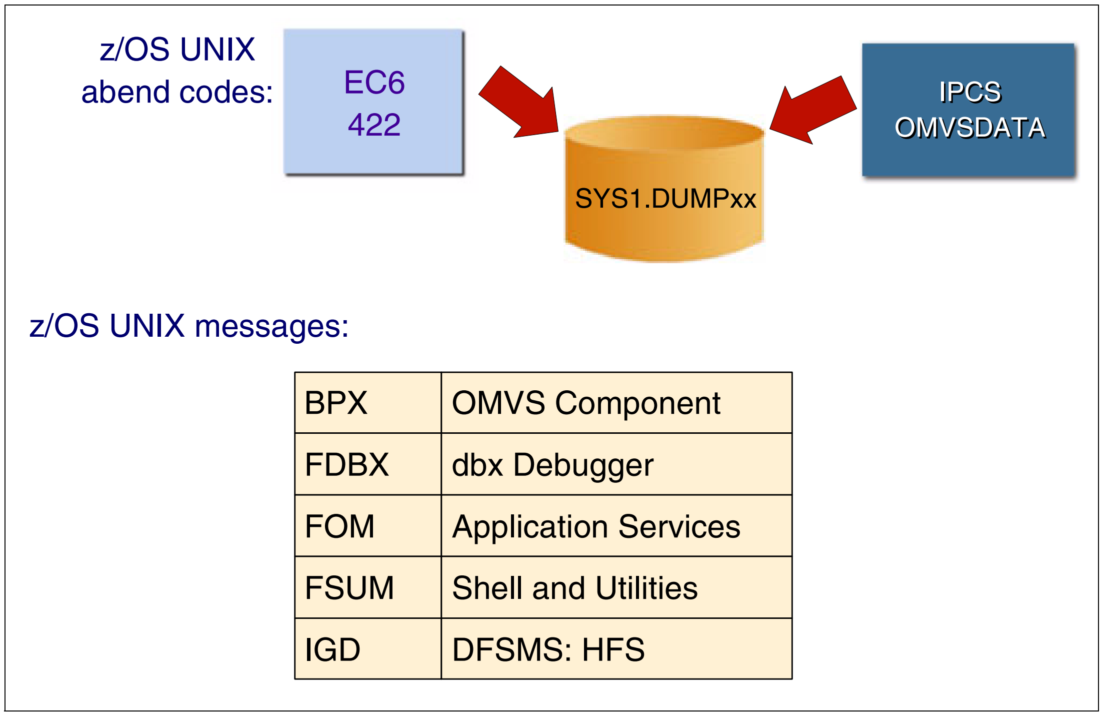
Figure 12-26 Where to find information to solve problems

If a problem occurs with z/OS UNIX System Services, the system writes an SVC dump and may issue messages. The SVC dump can be formatted and analyzed using IPCS. Messages referring to problems with z/OS UNIX kernel services will have the prefix `BPX`, shell messages will have the prefix `FSUM`, dbx messages will have the prefix `FDBX`, and messages relating to the hierarchical file system will have the prefix `GFU`. Problems with z/OS UNIX can be analyzed using the IPCS OMVSDATA keyword.

Problems with the z/OS UNIX shell and z/OS UNIX dbx are treated as application problems. If a SYSMDUMP data set is allocated for the TSO/E session, the system will create a core dump in an HFS file. The core dump must be copied to an MVS data set, and the dump can be formatted and analyzed using IPCS.

For debugging reasons, it is important to have a powerful trace facility. The **`CTRACE`** function in z/OS provides support for z/OS UNIX. The resulting trace data can be analyzed using IPCS.

12.27 z/OS UNIX ABENDs and messages



BPX	OMVS Component
FDBX	dbx Debugger
FOM	Application Services
FSUM	Shell and Utilities
IGD	DFSMS: HFS

Figure 12-27 Understanding z/OS UNIX ABENDs and messages

If there is a problem in the z/OS UNIX shell or debugger, the system will treat it as an application program. If a shell user has allocated a SYSMDUMP data set for the TSO/E session, the system will write a core dump in the user's working directory called coredump.pid.

In the example from stopping z/OS UNIX, we saw that `abend 422` was issued for the BPXOINIT process. This is normal.

All 422 abends and some EC6 abends may not be accompanied by an SVC dump because the IBM-supplied IEASLP00 parmlib member contains SLIP commands to suppress the dumps.

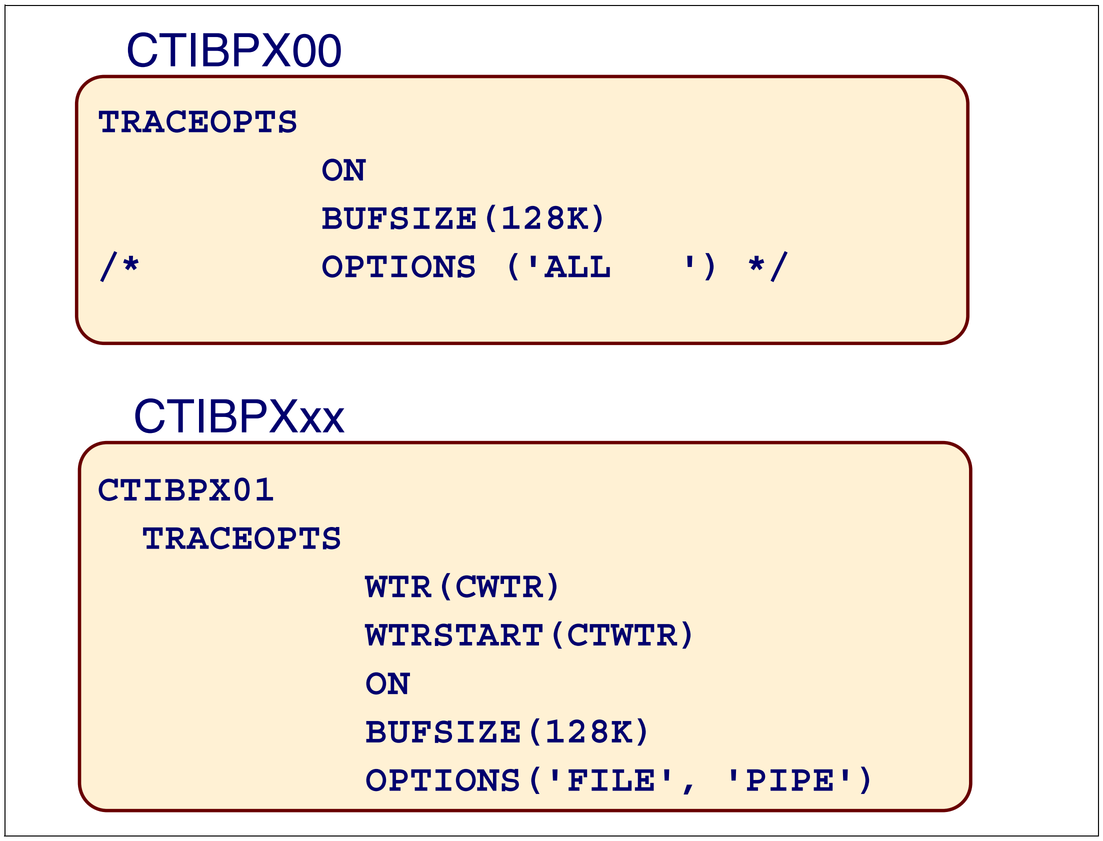
The IPCS command **`OMVSDATA`** can be used to analyze a dump from z/OS UNIX.

The abend codes are documented in *z/OS MVS System Codes,* SA22-7626. Reason codes can be found in an appendix of *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803.

Messages with the following prefixes are issued from z/OS UNIX:

- **FDBX** and **FSUM** - z/OS UNIX dbx and shell messages. For an explanation of the messages, *z/OS UNIX System Services Messages and Codes*, SA22-7807.
- **IGD** - DFSMS messages for the hierarchical file system.

12.28 CTIBPX00 and CTCBPXxx



CTIBPX00

```
TRACEOPTS
          ON
          BUFSIZE(128K)
/*        OPTIONS ('ALL   ') */
```

CTIBPXxx

```
CTIBPX01
  TRACEOPTS
            WTR(CWTR)
            WTRSTART(CTWTR)
            ON
            BUFSIZE(128K)
            OPTIONS('FILE', 'PIPE')
```

Figure 12-28 SYS1.PARMLIB members for tracing z/OS UNIX

Activating a CTIBPXxx parmlib member with additional trace options specified should only be done in situations with problems. Additional trace options will collect more information about z/OS UNIX and this can slow down the system performance noticeably. When requesting additional data to be traced, the trace buffersize should be increased. This cannot be done with **TRACE CT** commands.

The MVS component trace (CTRACE) provides support for z/OS UNIX. By default, z/OS UNIX performs a minimal amount of tracing at all times. The default tracing records only unusual events in z/OS UNIX to provide trace data when a problem occurs without slowing system performance. An installation can trace additional events by specifying tracing options in a CTIBPXxx parmlib member.

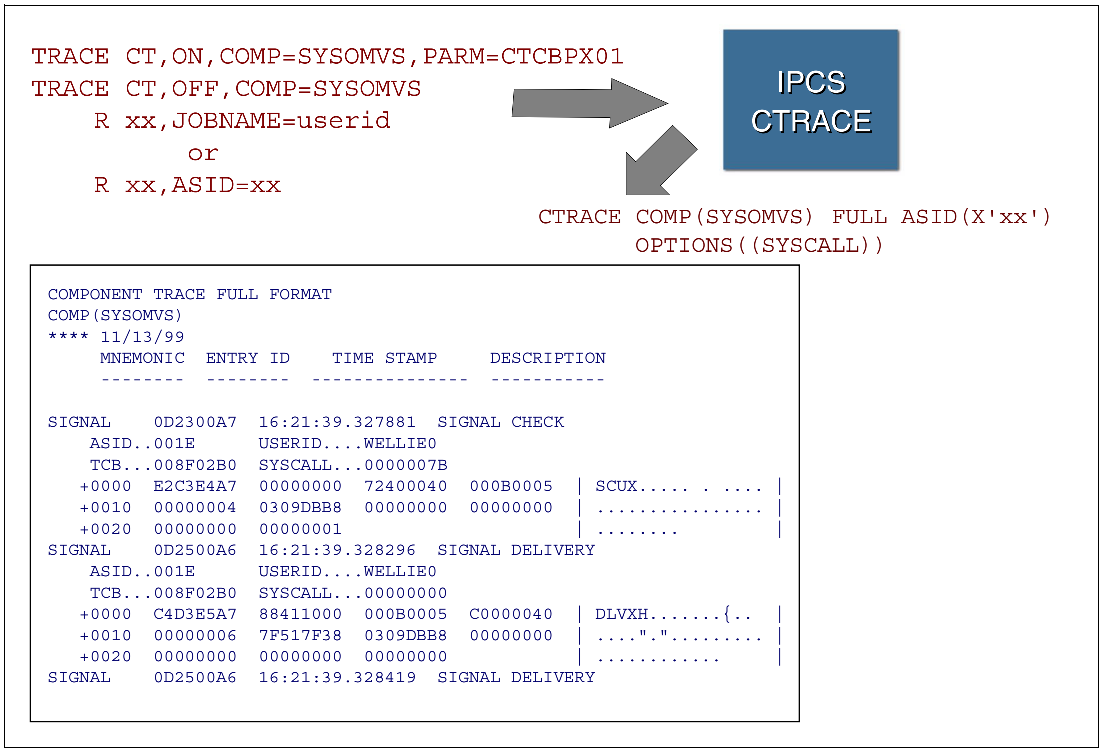
The system collects trace entries in a buffer. To analyze these entries, the buffer must be dumped to a data set. The installation can also decide to write the trace entries to a data set by using the component trace external writer.

The trace options are activated when z/OS UNIX is started. Trace options can be changed while the system is running by using the **TRACE CT** command. The trace options are activated by using IPCS to format the data. Trace data can be located in either:

- A buffer in an SVC or standalone dump
- A trace data set

You can change the bufsize dynamically.

12.29 Tracing z/OS UNIX events



```
TRACE CT,ON,COMP=SYSOMVS,PARM=CTCBPX01
TRACE CT,OFF,COMP=SYSOMVS
    R xx,JOBNAME=userid
          or
    R xx,ASID=xx
```

```
CTRACE COMP(SYSOMVS) FULL ASID(X'xx')
      OPTIONS((SYSCALL))
```

```
COMPONENT TRACE FULL FORMAT
COMP(SYSOMVS)
**** 11/13/99
     MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
     --------  --------  ---------------  -----------

SIGNAL    0D2300A7  16:21:39.327881  SIGNAL CHECK
    ASID..001E      USERID....WELLIE0
    TCB...008F02B0  SYSCALL...0000007B
   +0000  E2C3E4A7  00000000  72400040  000B0005  | SCUX..... . .... |
   +0010  00000004  0309DBB8  00000000  00000000  | ................ |
   +0020  00000000  00000001                      | ........         |
SIGNAL    0D2500A6  16:21:39.328296  SIGNAL DELIVERY
    ASID..001E      USERID....WELLIE0
    TCB...008F02B0  SYSCALL...00000000
   +0000  C4D3E5A7  88411000  000B0005  C0000040  | DLVXH.......{..  |
   +0010  00000006  7F517F38  0309DBB8  00000000  | ....".".......... |
   +0020  00000000  00000000  00000000            | ............     |
SIGNAL    0D2500A6  16:21:39.328419  SIGNAL DELIVERY
```

Figure 12-29 Commands to trace z/OS UNIX

To provide problem data, events are traced by the z/OS UNIX MVS component trace. When z/OS UNIX MVS starts, the trace automatically starts. The trace cannot be completely turned off while z/OS UNIX is running. The size of the trace buffers are specified in the parmlib member CTnBPXxx which is used by z/OS UNIX. The trace buffers require an IPL to be changed. They can be from 16 KB to 4 MB. The following commands control tracing:

- Operator can stop most tracing with the command:

  ```
  TRACE CT,OFF,COMP=SYSOMVS
  ```

- To change the CTnBPXxx parmlib member:

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

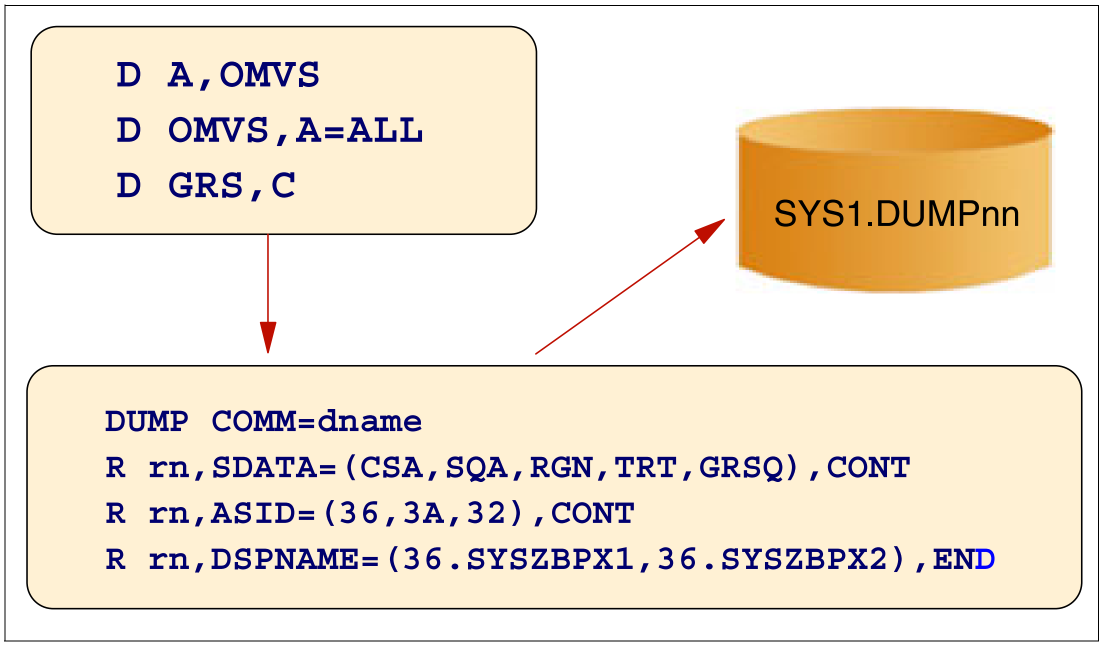
- To display information about a trace:

  ```
  DISPLAY TRACE,COMP=SYSOMVS
  ```

Use a user ID or address space ID to filter the trace. The IPCS **CTRACE** subcommand can be used to analyze trace records in a dump.

The previous figure showed the default CTIBPX00 parmlib member which starts minimal tracing. An installation can define other members with trace options which can be started if a problem occurs with z/OS UNIX. The trace buffer size cannot be changed while z/OS UNIX is active. To change buffer size, update a CTnBPXxx member with the new BUFSIZE value, stop z/OS UNIX, and restart z/OS UNIX using the new CTnBPXxx parmlib member.

12.30 Debugging a z/OS UNIX problem



```
D A,OMVS
D OMVS,A=ALL
D GRS,C
```

```
DUMP COMM=dname
R rn,SDATA=(CSA,SQA,RGN,TRT,GRSQ),CONT
R rn,ASID=(36,3A,32),CONT
R rn,DSPNAME=(36.SYSZBPX1,36.SYSZBPX2),END
```

Figure 12-30 Commands to debug z/OS UNIX and take an ABEND

Information about the kernel address space and its associated data spaces can be obtained from the MVS displays, as we have already seen.

When you need a dump to debug a problem, use the **DUMP** command to tell MVS what to dump. The dump will go to the SYS1.DUMPnn data set.

You need to dump several types of data in order to diagnose a problem, as follows:

- The OMVS kernel address space
- Any OMVS data spaces that may be associated with the problem
- Any OMVS process address spaces that may be associated with the problem
- Appropriate storage areas containing system control blocks

The sample dump command shows dumping the kernel address space, the appropriate user address spaces, and the two major kernel data spaces. You will need to allocate a very large dump data set, since you are dumping multiple address spaces and data spaces.

The dump title can be up to 100 characters. If you end each line with `CONT`, then the program will prompt you for more input data. When you are finished, code `END`. You will need to use the OMVS displays to determine the appropriate data to dump. After the dump completes, you receive an `IEA911E` message indicating whether the dump was complete.

In Figure 12-30, ASID=36 is the OMVS kernel. Address spaces 3A and 32 are other address spaces believed to be part of the problem. Refer to the dataspaces by their names with the kernel address space ID as a preface.

12.31 IPCS OMVSDATA reports

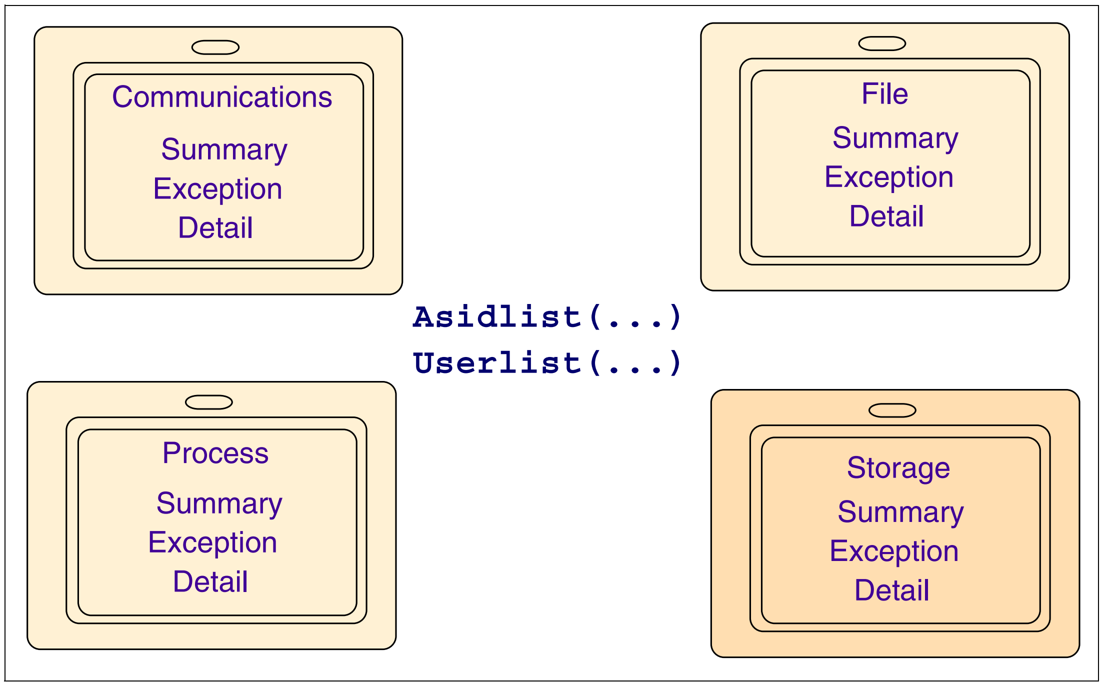


Figure 12-31 The OMVSDATA keyword in IPCS for analyzing problems

The IPCS OMVSDATA keyword provides support for analyzing problems with the z/OS UNIX component. SVC dumps or standalone dumps can be formatted using OMVSDATA. The following report types can be created:

- **Communications**: Provides information about z/OS UNIX pseudoterminal user connections.
- **File**: Provides information about each z/OS UNIX file system type and its mounted file systems.
- **Process**: Provides information about z/OS UNIX processes.
- **Storage**: Provides information about z/OS UNIX storage manager cell pools.

The default report type is PROCESS. For each report type, the level of detail can be determined by specifying: summary, exception, and detail. For each report, one or more of the filtering keywords can be used to limit the amount of data in the report, as follows:

- **ASIDLIST(asidlist):** requests that information be included for the ASIDs listed
- **USERLIST(userlist):** requests that information be included for the user IDs listed.

For an application dump (coredump.pid in the HFS), you will have to copy it out to a sequential data set. Then use IPCS to analyze it using the **`STATUS`** and **`SUMMARY FORMAT CURRENT`** subcommands. The OMVSDATA reports will not work on the application dump as they format system areas in the kernel.

13

z/OS UNIX shell and programming tools

This chapter introduces you to various programming environments, and highlights some of the possibilities for programming under z/OS UNIX.

It describes how to:

- Understand the environment needed
- Choose the most appropriate programming language
- Install the environment needed for programming
- Write short programs in different programming languages

13.1 Language Environment runtime library

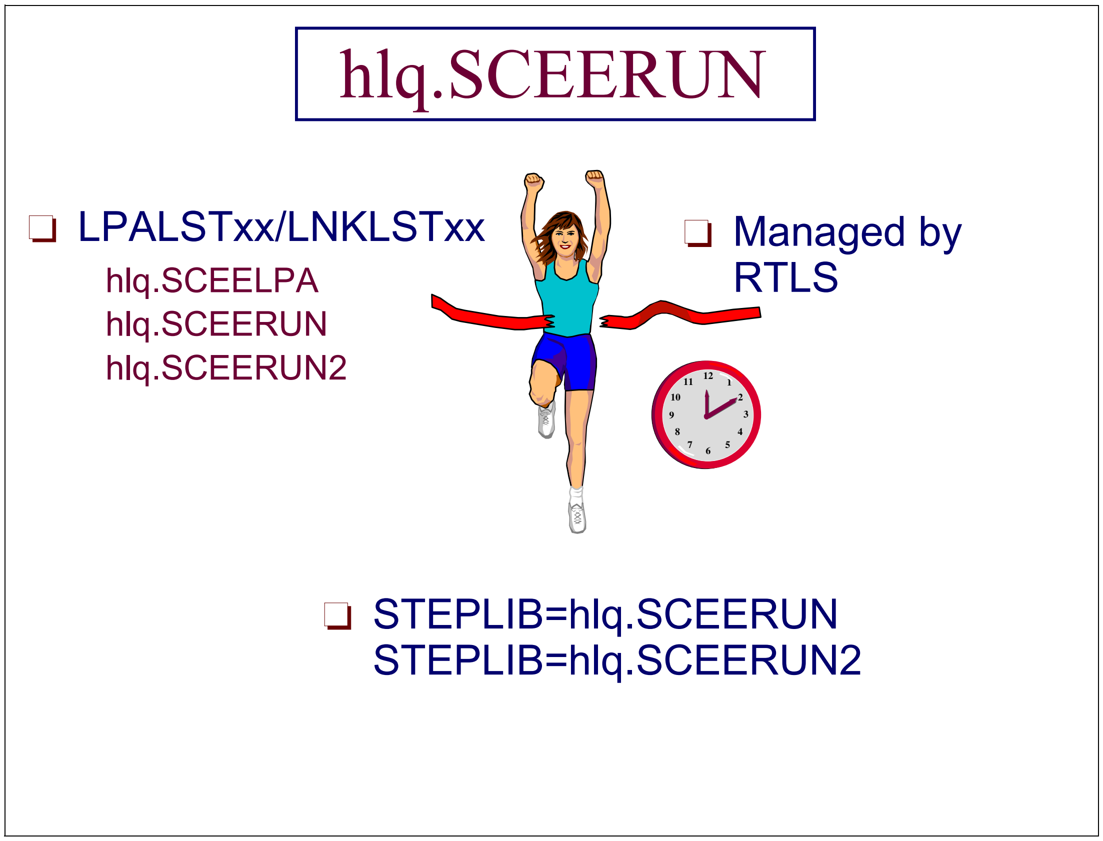


Figure 13-1 The LE run-time library specifications

For most z/OS UNIX applications, the Language Environment (LE) runtime library is needed for execution and comes from the SCEERUN data set. On average, about 4 MB of the runtime library are loaded into memory for every address space running Language Environment-enabled programs, and copied on every fork.

In OS/390 V2R10, a new data set, SCEERUN2, was added that contains LE load modules that are required to reside in a PDSE. Add this data set to the linklist. The SCEERUN and SCEERUN2 data sets can be:

- Placed in LPA/LNKLST

  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.

- Accessed via STEPLIB

  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. Note that this may affect performance somewhat.

- Managed by Runtime Library Services (RTLS)

When choosing a method for runtime library access, you should consider the following:

- Can the Language Environment runtime library be placed in LNKLST without adversely affecting other applications?
- Is the Language Environment runtime library heavily used at your installation?

- Does the RTL require frequent testing or replacement with new versions?

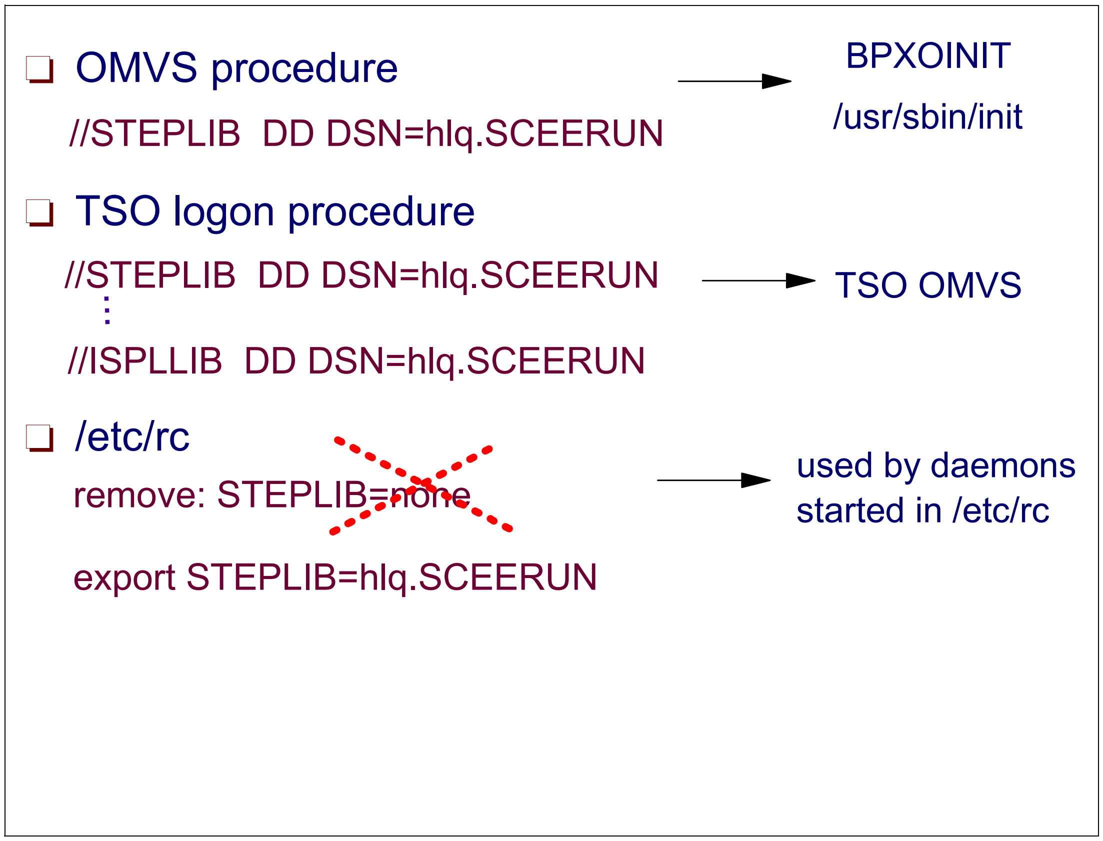
Some installations cannot put the current level of the LE runtime library into the LINKLIST because older Language Environment levels are needed to run key production applications. This means that key runtime library routines cannot be put in the LPA for better performance. In addition, you cannot put the SCEELPA data set as part of the LPALSTxx.

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, as follows:

- For the current z/OS C/C++ compiler:
  - If you are using the z/OS shell, issue the following command:
    ```
    export STEPLIB="CBC.SCBCCMP"
    ```
  - If you are using the tcsh shell, issue the following command:
    ```
    setenv STEPLIB "CBC.SCBCCMP"
    ```
- 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(TM) 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++ utility.

13.2 Placing SCEERUN in the STEPLIB



OMVS procedure
//STEPLIB DD DSN=hlq.SCEERUN
BPXOINIT
/usr/sbin/init
TSO logon procedure
//STEPLIB DD DSN=hlq.SCEERUN
TSO OMVS
//ISPLLIB DD DSN=hlq.SCEERUN
/etc/rc
remove: STEPLIB=none
used by daemons started in /etc/rc
export STEPLIB=hlq.SCEERUN

Figure 13-2 Placing the SCEERUN data set in the proper place

Sometimes the SCEERUN data set cannot be placed in LNKLST, because other applications require the pre-Language Environment run-time libraries. In that case, you can make the Language Environment run-time library available through STEPLIB. In addition, you can use this approach to test new levels of the run-time libraries. Perform the following steps:

- STEPLIB OMVS procedure

  Add the SCEERUN data set on a STEPLIB DD statement to the OMVS startup procedure found in PROCLIB. This will cause the STEPLIB data set to be propagated to BPXOINIT and the /usr/sbin/init program including all programs it invokes using fork or exec.

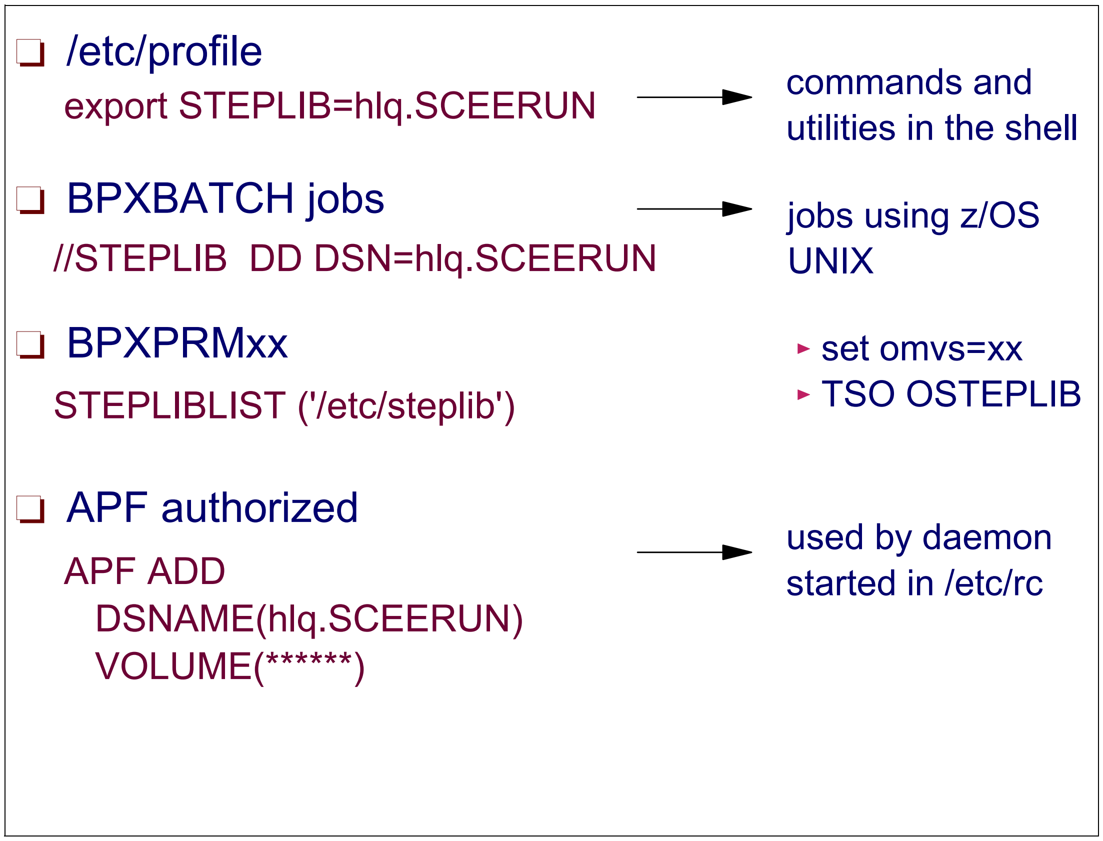
- TSO/E logon

  Add the SCEERUN data set to your TSO/E logon procedure, by concatenating it to the ISPLLIB DD statement and concatenating it to the STEPLIB DD statement. The TSOLIB function may also be used. This will be used when issuing the TSO/E **OMVS** command.

- /etc/rc

  Add the statement to the /etc/rc file. This will be used by daemons started in /etc/rc.

13.3 Placing SCEERUN in the STEPLIB (2)



- /etc/profile
  export STEPLIB=hlq.SCEERUN → commands and utilities in the shell
- BPXBATCH jobs → jobs using z/OS UNIX
  //STEPLIB DD DSN=hlq.SCEERUN
- BPXPRMxx
  STEPLIBLIST ('/etc/steplib')
  - set omvs=xx
  - TSO OSTEPLIB
- APF authorized
  APF ADD DSNAME(hlq.SCEERUN) VOLUME(******) → used by daemon started in /etc/rc

Figure 13-3 Where to place the SCEERUN data set

- /etc/profile

  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.

- STEPLIB BPXBATCH

  Add the SCEERUN data set on a STEPLIB DD statement to any job invoking BPXBATCH.

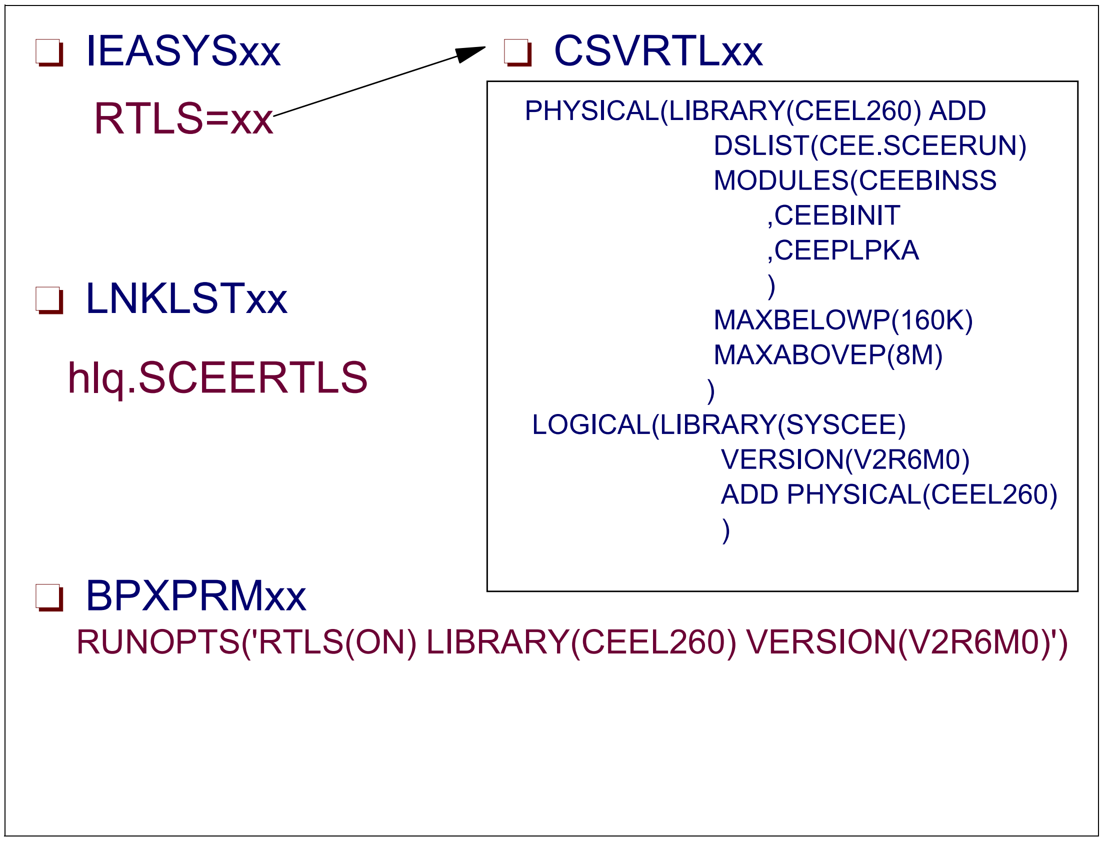
- STEPLIBLIST

  Add the SCEERUN data set to the STEPLIBLIST facility of the BPXPRMxx PARMLIB member.

- APF Authorization

  The SCEERUN data set will also need to be APF-authorized.

13.4 Managing RTL with RTLS



```
PHYSICAL(LIBRARY(CEEL260) ADD
                          DSLIST(CEE.SCEERUN)
                          MODULES(CEEBINSS
                                 ,CEEBINIT
                                 ,CEEPLPKA
                                 )
                          MAXBELOWP(160K)
                          MAXABOVEP(8M)
                         )
 LOGICAL(LIBRARY(SYSCEE)
                          VERSION(V2R6M0)
                          ADD PHYSICAL(CEEL260)
                          )
```

Figure 13-4 Managing the RTL with the RTLS

Some installations cannot put the current level of the LE runtime library into the LINKLIST because older LE levels are needed to run key production applications. This means that key run-time library routines cannot be put in the LPA for better performance. In addition, you cannot put the SCEELPA data set as part of the LPALSTxx.

The answer to this problem is Run-Time Library Services (RTLS). RTLS enables you to eliminate STEPLIBs from the JCL that runs your applications. By eliminating STEPLIBs, you reduce the installation management your application requires, as well as the system overhead involved in searching STEPLIB data sets when loading modules into storage. In place of STEPLIBs, the CSVRTLS macro connects to and loads from a given RTLS logical library.

With RTLS you can put key runtime library modules from more than one level of Language Environment into common storage for shared access.

Language Environment applications currently can exploit RTLS by using the Language Environment run-time options RTLS(ON), LIBRARY(le_run-time_lib) and VERSION(version), which identify the RTLS logical library to be connected.

If you want to use the BPXBATCH jobs, you must consider that the RUNOPTS will not be passed to BPXBATCH. If you want to use the same set of Run-Time Libraries, you must customize the BPXBATCH job as the following JCL shows and add the //STDENV DD... or the //STEPLIB DD... statement:

```
//GGIBPXBA   JOB   (55,500,,999),'MVS      ',CLASS=A,
// MSGCLASS=R,REGION=OK,NOTIFY=&SYSUID
//LSBPXB   EXEC PGM=BPXBATCH,REGION=8M,
//   PARM='SH ls /usr/lib'
//*STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//STDOUT   DD  PATH='/u/ggi/bin/mystd.out',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//STDERR   DD  PATH='/u/ggi/bin/mystd.err',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//STDENV   DD  *
_CEE_RUNOPTS=RTLS(ON) LIBRARY(xxxxxx) VERSION(xxxxxx)
//
```

Remember that the z/OS CS must also be updated with the RUNOPTS statements (TCPIP, FTPD, and so on).

13.5 Overview of c89/cc/c++

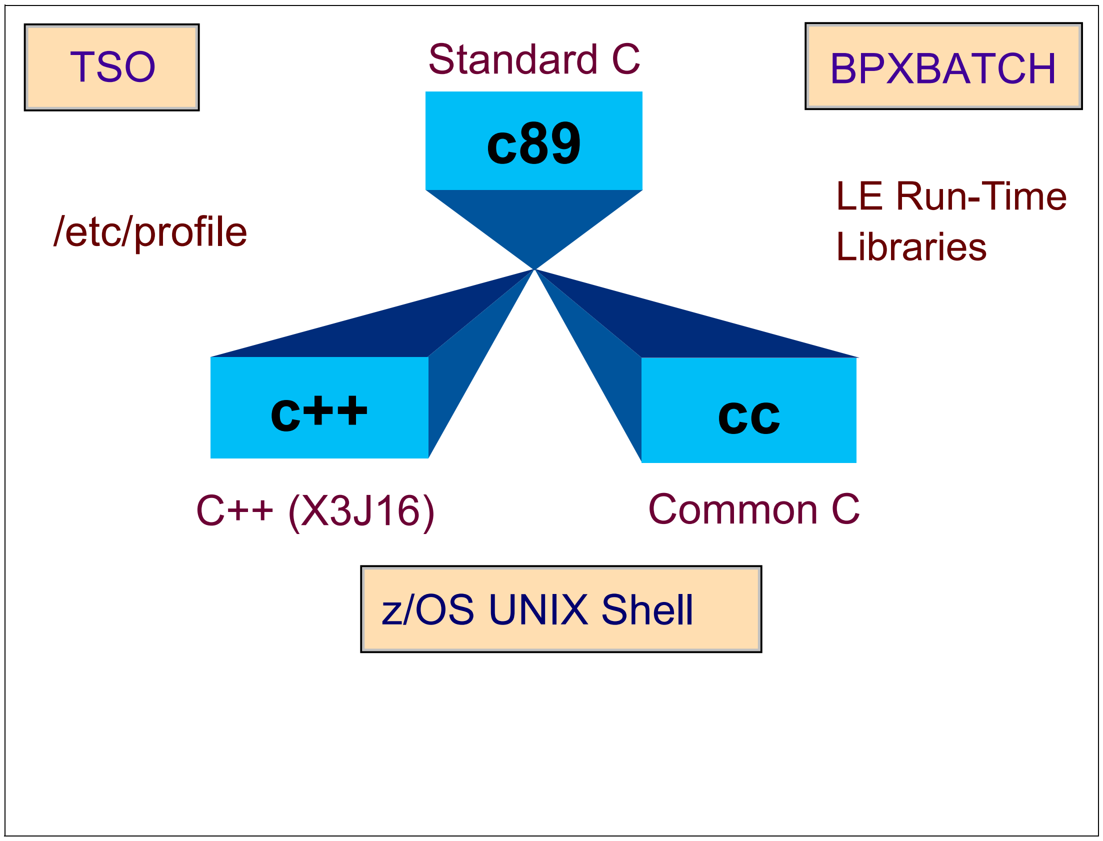


Figure 13-5 Overview of the C/C++ functions with z/OS UNIX

z/OS UNIX can be used from C/C++ functions and assembler calls. Applications written in COBOL or PL/I can use z/OS UNIX indirectly through C/C++ functions or assembler calls. If you must use a previous level of the compiler, or target the executables produced by c89 to run on a previous level of the run-time library, then you must customize other environment variables.

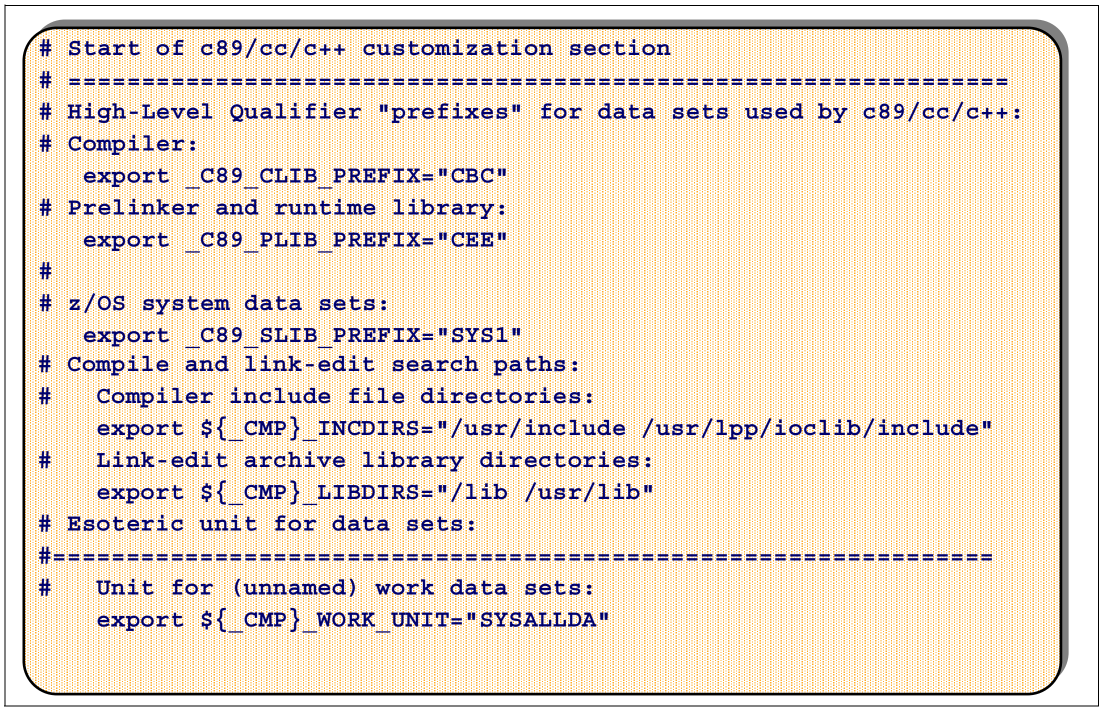
c89, cc, and c++ compile, assemble, and link-edit z/OS C and z/OS C++ programs, as follows:

- c89 should be used when compiling C programs that are written according to Standard C.
- cc should be used when compiling C programs that are written according to Common Usage C.
- c++ must be used when compiling C++ programs, which are written according to Draft Proposal International Standard for Information Systems—Programming Language C++ (X3J16). c++ can compile both C++ and C programs, and can also be invoked by the name cxx.

The c89 utility is customized by setting environment variables. The ones that most commonly require setting are specified in the c89 customization section in /etc/profile.

The customization section in /etc/profile assumes that you are using the current level of z/OS C/C++ compiler and Language Environment runtime library.

13.6 Customization of /etc/profile for c89/cc/c++



```
# Start of c89/cc/c++ customization section
# =================================================================
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# Compiler:
   export _C89_CLIB_PREFIX="CBC"
# Prelinker and runtime library:
   export _C89_PLIB_PREFIX="CEE"
#
# z/OS system data sets:
   export _C89_SLIB_PREFIX="SYS1"
# Compile and link-edit search paths:
#    Compiler include file directories:
     export ${_CMP}_INCDIRS="/usr/include /usr/lpp/ioclib/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="SYSALLDA"
```

Figure 13-6 Customization of /etc/profile for cc/c++

If c/c++, LE, or z/OS do 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 Figure 13-6 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.

In order to get the cxx environment up and running, you have to customize the /etc/profile.

The environment variables used by the cc utility have the same names as the ones used by c89, except that the prefix is _CC instead of _C89. Likewise, for the c++ (cxx) utility, the prefix is _CXX instead of _C89. Normally, you do not need to explicitly export the environment variables for all three utilities; the `eval` commands at the bottom of the c89 customization section of the sample /etc/profile can be used. These commands set the variables for the other utilities, based on those set for c89.

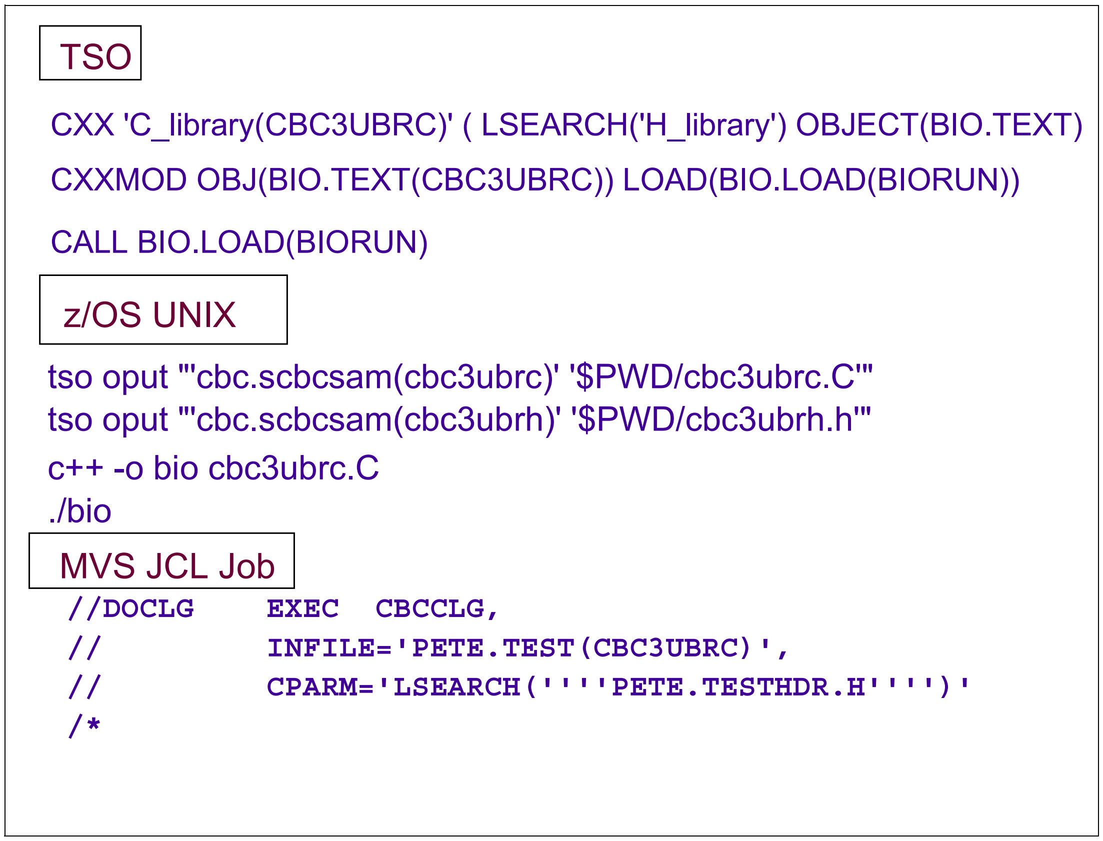
Customization for /etc/profile: By placing any customization statements for c89 into /etc/profile and uncommenting those lines, the environment variables will automatically be exported through the `eval` command for cc and c++ as well.

Example 13-1 on page 514 shows you an environment with the correct settings of the /etc/profile c89/cc/c++ section. In this example, the RTLS was implemented. You can see this in the _CEE_RUNOPTS statement.

Example 13-1

```
GGI:TC4:/u/ggi:==> env
  ....
  _CEE_RUNOPTS=RTLS(ON) LIBRARY(SYSCEE) VERSION(V1R6M0)
  COLUMNS=80
_CC_PLIB_PREFIX=CEE
  _=/bin/env
  LOGNAME=GGI
  STEPLIB=none
  ENV=/u/ggi/.setup
  LANG=C
  LIBPATH=/lib:/usr/lib:.
_CXX_SLIB_PREFIX=SYS1
  _CXX_CLIB_PREFIX=CBC
  _C89_LIBDIRS=/lib /usr/lib
  TERM=dumb
_C89_WORK_UNIT=SYSDA
  _C89_INCDIRS=/usr/include
  HOME=/u/ggi
_CC_SLIB_PREFIX=SYS1
  LINES=28
_CC_CLIB_PREFIX=CBC
  _C89_PLIB_PREFIX=CEE
  TZ=EST5EDT
  MANPATH=/usr/man/%L
  NLSPATH=/usr/lib/nls/msg/%L/%N
  GGI:TC4:/u/ggi:==>
```

13.7 Compile, link-edit, and run



TSO

```
CXX 'C_library(CBC3UBRC)' ( LSEARCH('H_library') OBJECT(BIO.TEXT)
CXXMOD OBJ(BIO.TEXT(CBC3UBRC)) LOAD(BIO.LOAD(BIORUN))
CALL BIO.LOAD(BIORUN)
```

z/OS UNIX

```
tso oput "'cbc.scbcsam(cbc3ubrc)' '$PWD/cbc3ubrc.C'"
tso oput "'cbc.scbcsam(cbc3ubrh)' '$PWD/cbc3ubrh.h'"
c++ -o bio cbc3ubrc.C
./bio
```

MVS JCL Job

```
//DOCLG     EXEC   CBCCLG,
//                 INFILE='PETE.TEST(CBC3UBRC)',
//                 CPARM='LSEARCH(''''PETE.TESTHDR.H'''')'
/*
```

Figure 13-7 Ways to compile, link-edit, and execute C/C++ programs

To compile, link-edit, and run under TSO:

- Compile:

```
CXX 'PETE.TEST.C(CBC3UBRC)' (LSEARCH('PETE.TESTHDR.H') OBJECT(BIO.TEXT)
```

- Prelink and link:

```
CXXMOD OBJ(BIO.TEXT(CBC3UBRC)) LOAD(BIO.LOAD(BIORUN))
```

- Run the program:

```
CALL BIO.LOAD(BIORUN)
```

Under z/OS UNIX, put the source into files in the HFS:

- Compile, prelink, and link:

```
c++ -o bio cbc3ubrc.C
```

- Run the program:

```
GGI:/u/ggi:==> ./bio
```

Under batch, use the cataloged procedure CBCCLG to compile, link, and run:

```
//DOCLG     EXEC  CBCCLG,
//          INFILE='PETE.TEST(CBC3UBRC)',
//          CPARM='LSEARCH(''''PETE.TESTHDR.H'''')'
/*
```

13.8 Customization of Java™ for z/OS

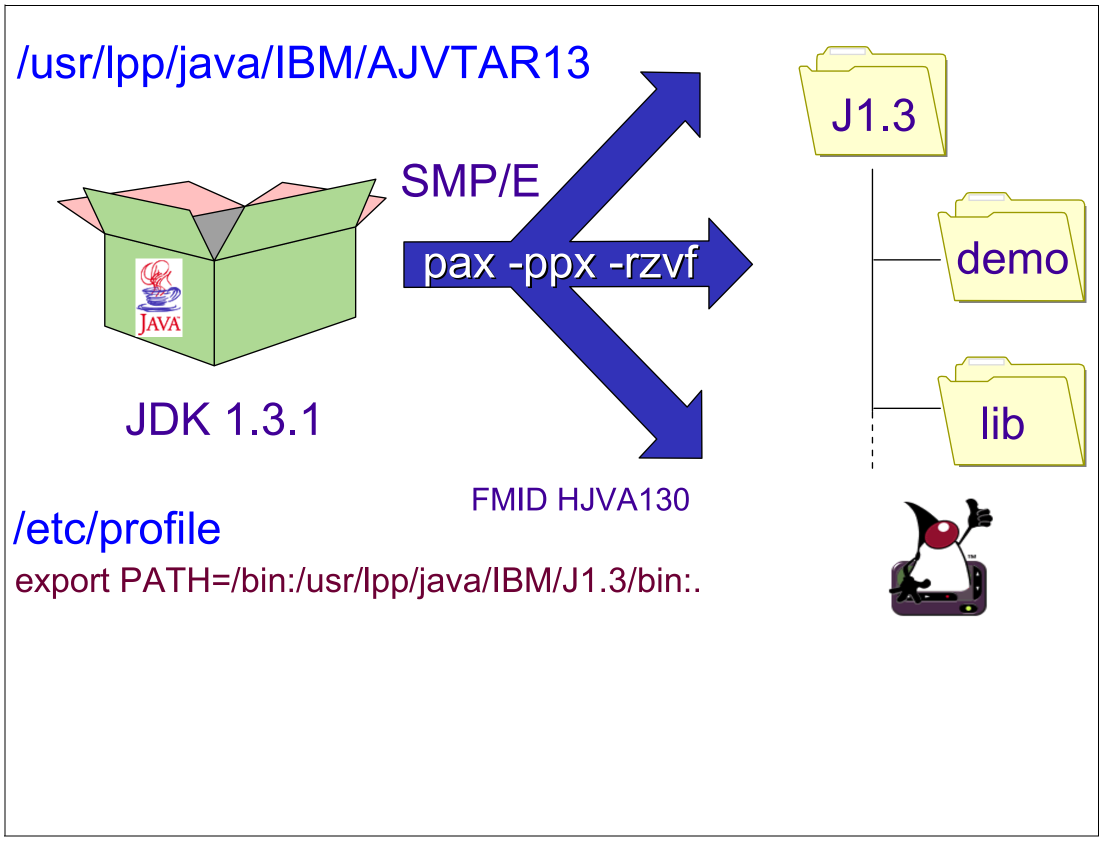

/usr/lpp/java/IBM/AJVTAR13
SMP/E
pax -ppx -rzvf
JDK 1.3.1
J1.3
demo
lib
FMID HJVA130
/etc/profile
export PATH=/bin:/usr/lpp/java/IBM/J1.3/bin:.

Figure 13-8 Customization of Java for z/OS

The Java for z/OS product is IBM's port of Sun Microsystems' Java Development Kit (JDK) to the S/390 platform. The Java for z/OS product at the JDK 1.3.1 level is certified as a fully compliant Java product.

Java for z/OS is a full object-oriented language. In contrast to C++, you have to write your programs with object-oriented code. (In C++, you do not have to use object-oriented code.) The big difference between the C++ code and Java code is that Java does not use pointer and pointer arithmetic.

Java for z/OS is operational within any version and release of the z/OS operating system. It provides a Java execution environment equivalent to that available on any other server platform.

Java Programming Language: Java Programs: Write once, run everywhere.

If ordered with ServerPac, the Java code has already been put in the root HFS under the path of /usr/lpp/java. The FMID for Java for is HJVA130.

To find the requested files for Java, you have to add the path /usr/lpp/java/IBM/J1.3/bin to the /etc/profile or $HOME/.profile.

13.9 Java virtual machine

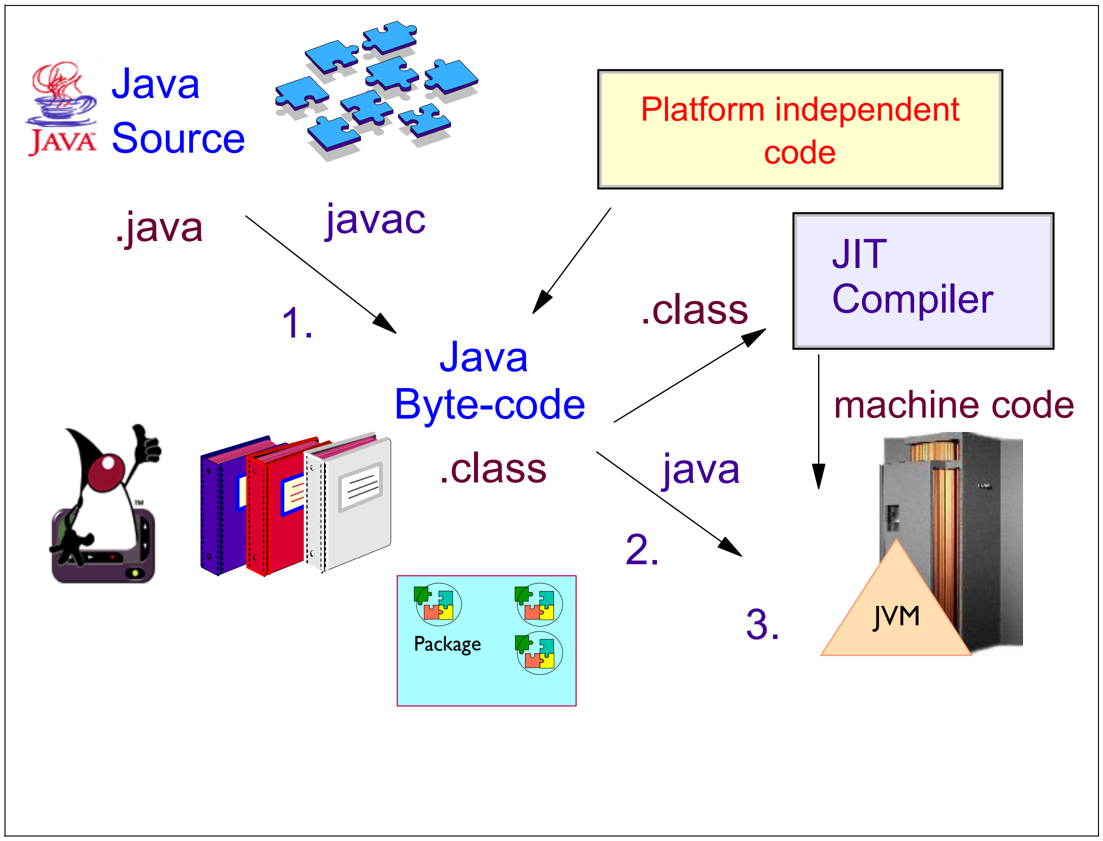


Figure 13-9 The Java Virtual Machine

A *Virtual Machine* is the processor on which Java's byte-codes run. It is the instruction set that the interpreter understands.

The compiler, javac, takes the Java source code and produces Java byte-codes. These byte-codes correspond to the language of the virtual machine. The file organization is such that byte-codes are per class, that is, one .class file per Java class definition.

These class files may be loaded across the network.

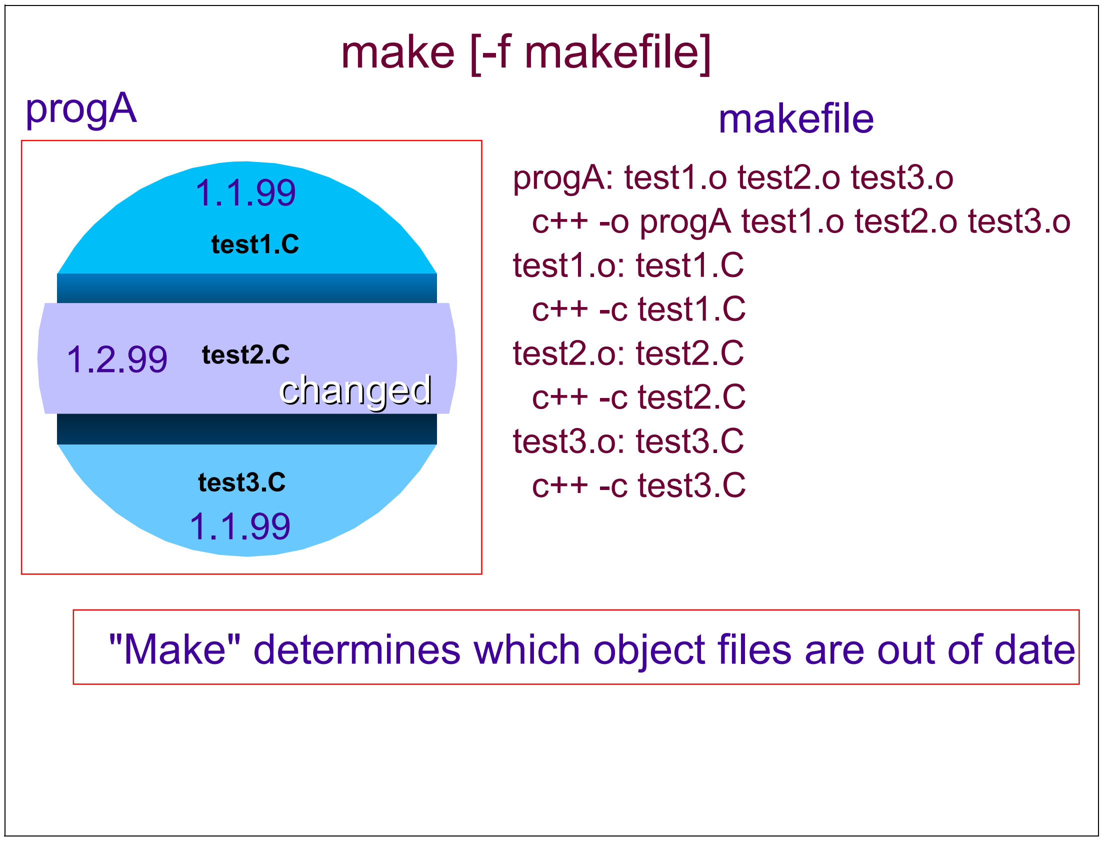
Because of the need for architecture independence, performance tuning must be performed on the client side. This client-side compilation is known as just-in-time (JIT) compilation.

Since the Java virtual machine (JVM) does not necessarily correspond to any particular hardware/operating system, the .class files are portable to any implementation of the virtual machine. This is the essence of Java's portability.

JIT will take Java byte-code and generate native code for the client processor. Many optimizations are possible that can lead to execution speeds which are competitive with C/C++ (within a factor of 2 to 5).

The JIT compiler is built specifically for z/OS. It provides execution time improvements over the interpreter. By default, the JIT compiler is activated but can be deactivated by setting an environment variable.

13.10 Management of software and the make utility



```
progA: test1.o test2.o test3.o
  c++ -o progA test1.o test2.o test3.o
test1.o: test1.C
  c++ -c test1.C
test2.o: test2.C
  c++ -c test2.C
test3.o: test3.C
  c++ -c test3.C
```

Figure 13-10 Using the make utility to manage software

Using the *make* utility can be a key factor in the successful management of software development projects, as well as any other type of project where you must keep a collection of files in synchronization with one another.

For example, suppose a program is built from several separate object files, each of which depends on its own source file. If you change a source file and then run make, make can automatically determine which object files are out of date (older than their corresponding source files).

The information that the make utility uses is in a file called the makefile. To invoke the utility, simply use **`make -f makefile`**, or, if you are in the same directory as the makefile, use only **`make`**.

This section introduces the syntax used in the makefile:

- Program: test1.o test2.o test3.o

  This tells the program that progA depends upon the three files with the names test1.o, test2.o and test3.o. If any or all of the .o files have changed since the last time the program was made, make attempts to remake the program. It does this using the recipe on the next line. This recipe consists of a C++ command that links programs from the three objects.

- c++ -o progA test1.o test2.o test3.o

  If the make utility determines the changes, it will execute this line and try to link the program from the three objects.

- test1.o: test1.C

  This statement tells the make utility that if the test1.C code has changed, it must recompile the code to the object test1.o with the following commands.
- c++ -c test1.C

  As mentioned previously, the make utility recompiles the test1.C code and creates the object test1.o.

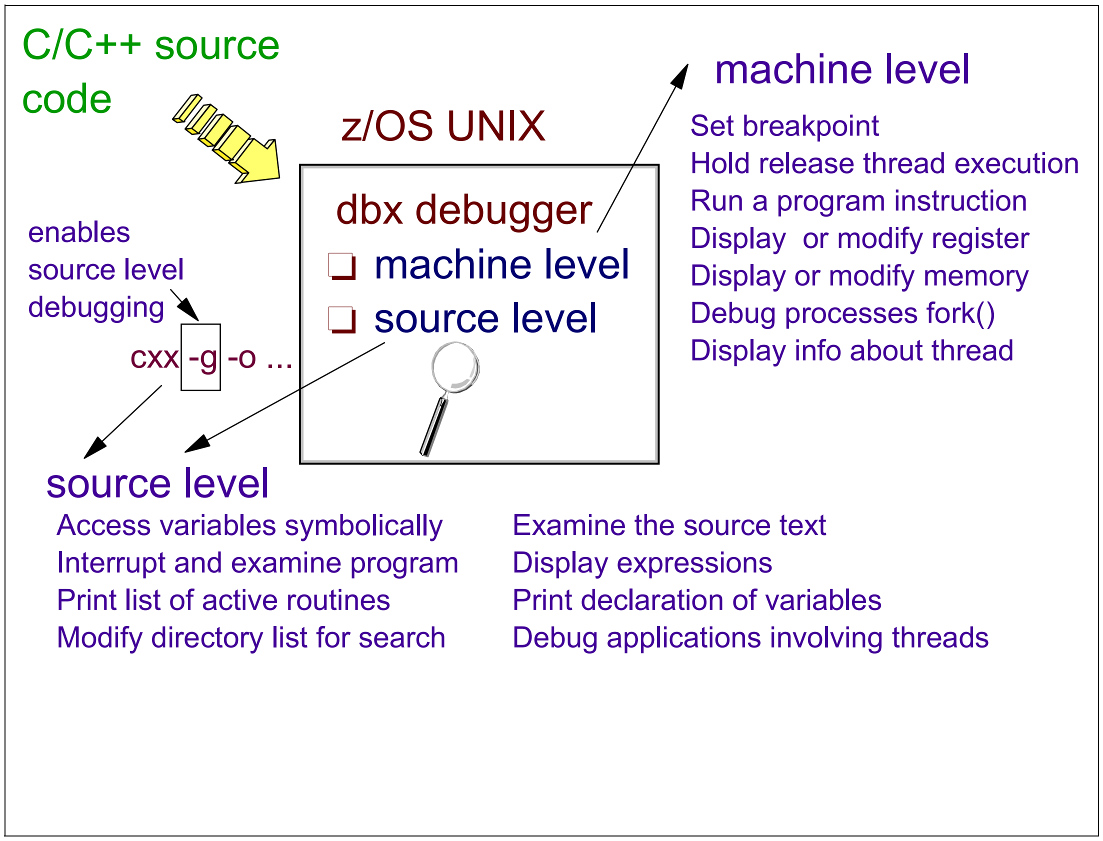
The next statements are the same as for the test1.C file.

The syntax of the make utility is:

```
target target ... : prerequisite
      prerequisite ... <tab> recipe
```

The <tab> syntax is a character with `X'05'`. This is not displayable from the ISPF Editor.

13.11 The dbx debugger



C/C++ source code
z/OS UNIX
machine level
Set breakpoint
Hold release thread execution
Run a program instruction
Display or modify register
Display or modify memory
Debug processes fork()
Display info about thread
dbx debugger
❑ machine level
❑ source level
enables source level debugging
cxx -g -o ...
source level
Access variables symbolically
Interrupt and examine program
Print list of active routines
Modify directory list for search
Examine the source text
Display expressions
Print declaration of variables
Debug applications involving threads

Figure 13-11 The dbx debugger

You need to create a z/OS UNIX C/MVS™ application program that will compile, link-edit, and run successfully. After your program has been developed, you can take advantage of the z/OS UNIX debugger (with its dbx utility) to debug the program from within the shell environment on an MVS system.

Using dbx, you can debug your program at the source level and at the machine level.

To trace the source level, the programs must be compiled with the -g option:

```
c89 -g -o looper looper.c
```

The dbx debugger also has some restrictions that must be considered before debugging programs. Some of them are:

- It can only debug key(8) programs
- It cannot debug programs in supervisor state
- It cannot source-level debug DLLs below OS390 V2R2.

13.12 Introduction to shells

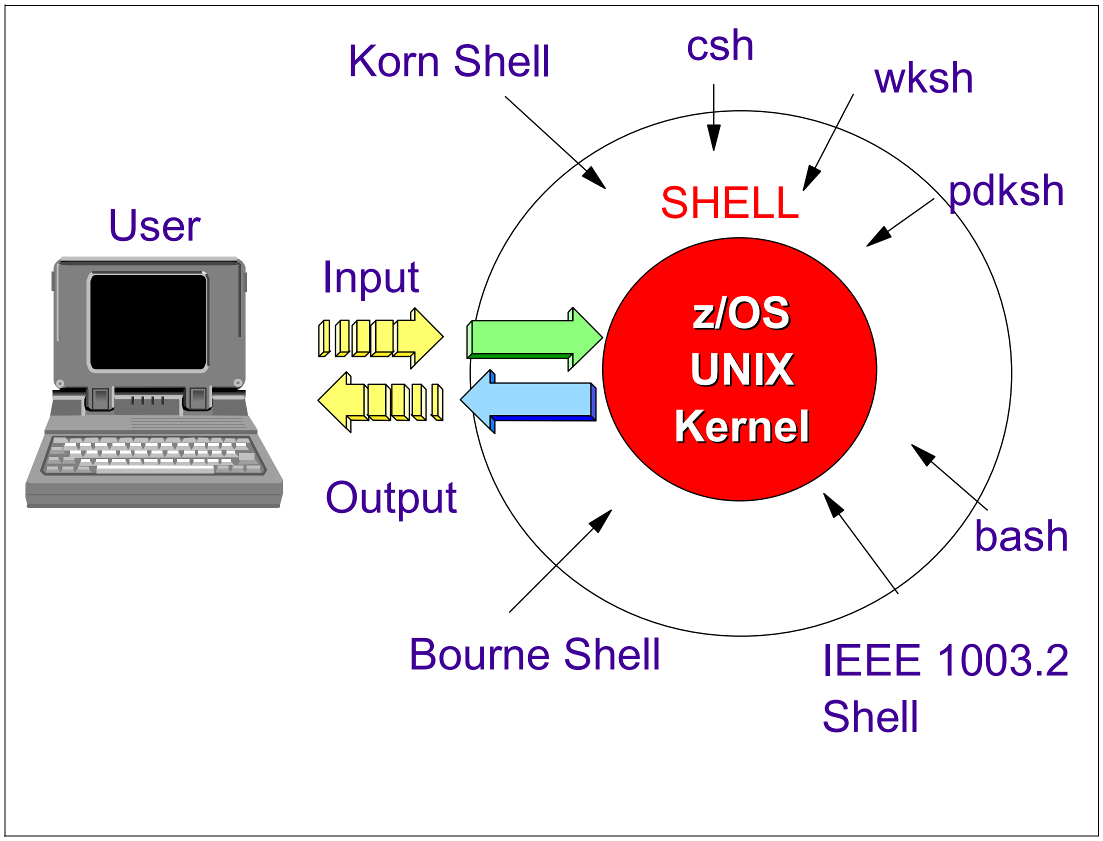


Figure 13-12 An introduction to the many shells available with z/OS UNIX

What is a *shell*? A shell's job is to translate the user's command lines into operating system instructions.

There are many different shells available for UNIX systems, as you can see. This led to the current situation, where multiplicity of similar software has led to confusion, lack of compatibility, and—most unfortunate of all—the inability of UNIX to capture as big a share of the market as other operating platforms.

The differences between the shells are minimal but have a big impact. Under z/OS UNIX, you can supply an alternate shell. The shells shown in Figure 13-12 are:

IEEE POSIX 1003.2	The standard used by z/OS UNIX
Bourne Shell	The first major shell; named after Steven Bourne (sh)
C shell	An alternative to the Bourne Shell, which was written at Berkley
Korn Shell	Compatible with the Bourne Shell; written by David Korn (ksh)
wksh	Windowing Korn Shell, which is the ksh shell with the extension for GUI
pdksh	Public Domain V7-based; written by Eric Gisin
bash	The Bourne-Again Shell; written by Brian Fox and Chet Ramey

You can find a short description of how to set up the Korn Shell in *z/OS UNIX System Services Planning*, GA-22-7800.

The steps are as follows:

1. Download the shell program, for example, Korn Shell from the z/OS UNIX home page.
2. Install to a directory like /ksh.
3. Consider turning on the sticky bit and putting the program in the Link Pack Area.
4. For users that want this shell as the default shell, change the PROGRAM in the OMVS RACF user profile as follows: /ksh/bin/ksh.
5. Customize /etc/init.options with the following statements:

   ```
   -sh /ksh/bin/ksh
   -e  SHELL=/ksh/bin/ksh
   ```

6. You can test the environment variables after changing the alternate shell with the command **env**:

   ```
   SHELL=/ksh/bin/ksh
   PATH=/ksh/bin:. ..........
   ```

13.13 REXX, CLISTs, and shell scripts

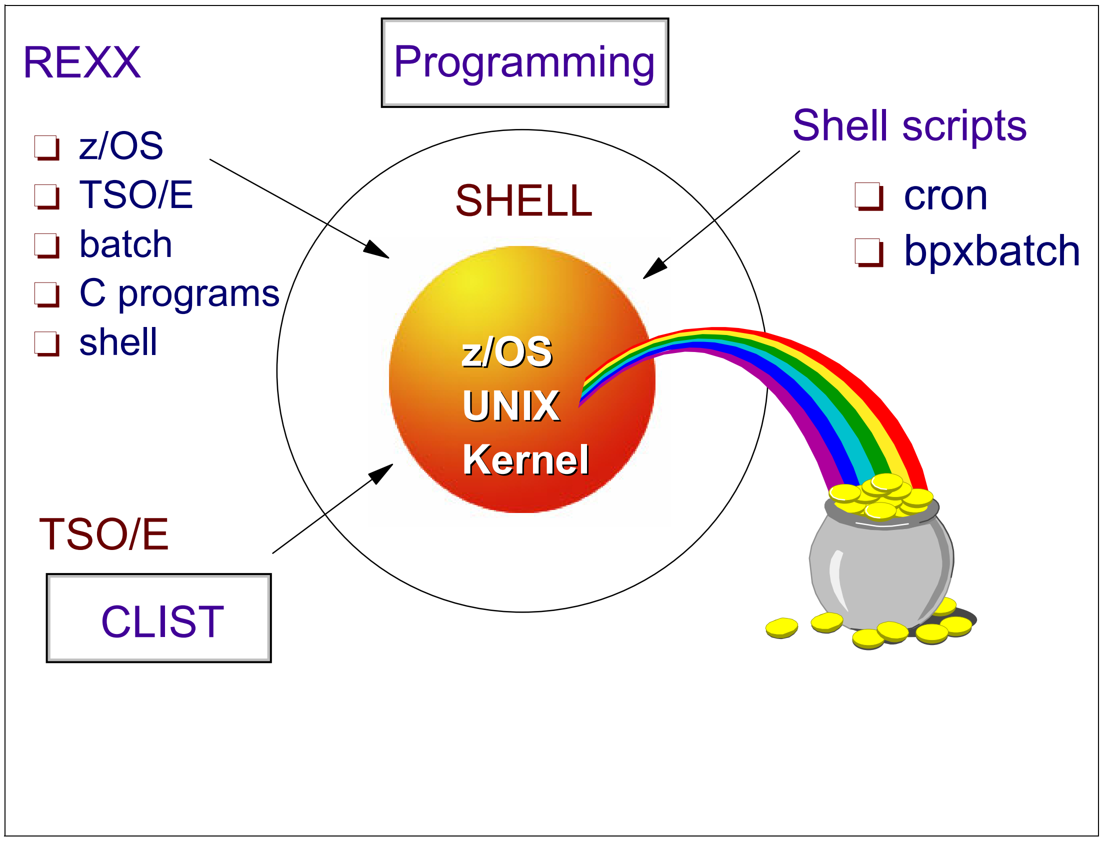


Figure 13-13 Different languages that can be used to create shell scripts

The shell programming environment with its shell scripts provides function similar to the TSO/E environment with its command lists (CLISTs) and the REstructured eXtended eXecutor (REXX) execs. The CLIST language is a high-level interpreter language that lets you work efficiently with TSO/E. A CLIST is a program, or command procedure, that performs a given task or group of tasks.

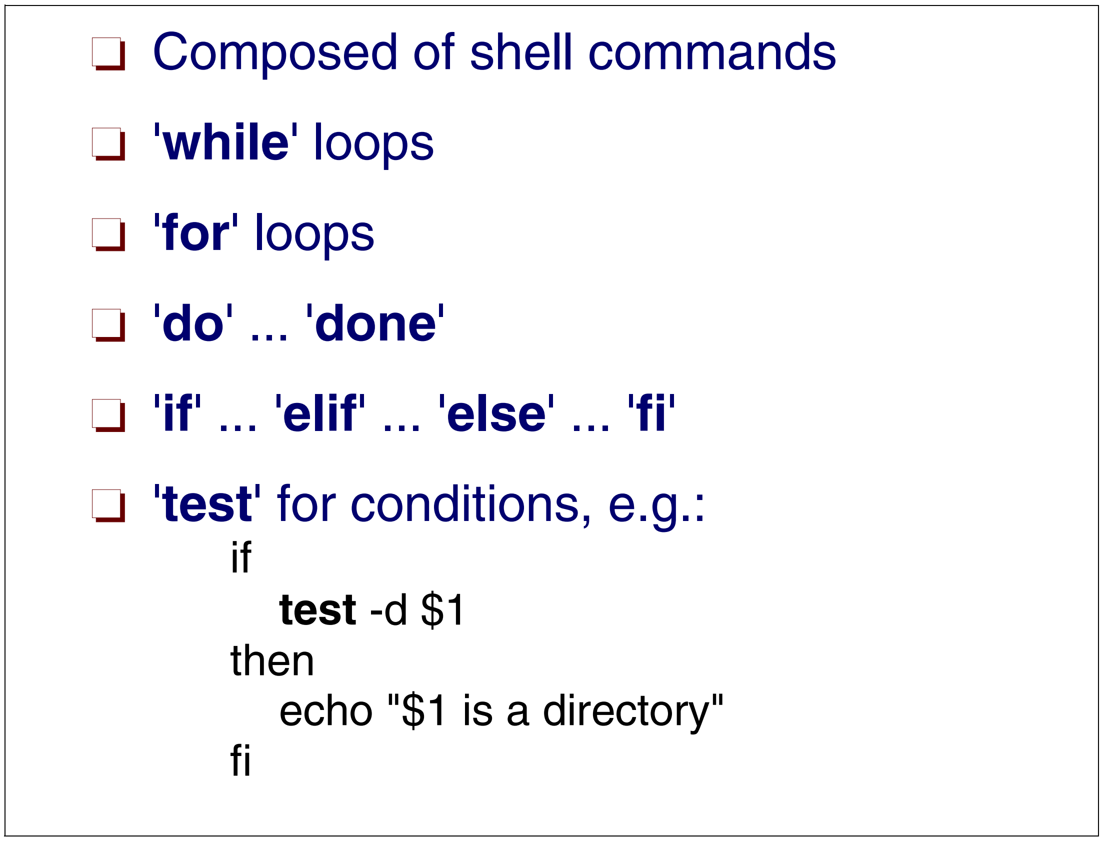
The REXX language is a high-level interpreter language that enables you to write programs in a clear and structured way. You can use the REXX language to write programs called REXX programs, or REXX execs, that perform given tasks or groups of tasks. REXX programs can run in any z/OS address space. You can run REXX programs that call z/OS UNIX services in TSO/E, in batch, in the shell environment, or from a C program.

In the z/OS shell, command processing is similar to command processing for CLISTs. You can write executable shell scripts (a sequence of shell commands stored in a text file) to perform many programming tasks. They can run in any dubbed z/OS address space. They can be run interactively, using cron, or using BPXBATCH. With its commands and utilities, the shell provides a rich programming environment.

Performance improvement: To improve performance when running shell scripts, add to the export statement in /etc/profile or $HOME/.profile:

```
   _BPX_SPAWN_SCRIPT=YES.
   _BPX_SHAREAS=YES
```

13.14 Shell script syntax



- Composed of shell commands
- '**while**' loops
- '**for**' loops
- '**do**' ... '**done**'
- '**if**' ... '**elif**' ... '**else**' ... '**fi**'
- '**test**' for conditions, e.g.:

```
if
    test -d $1
then
    echo "$1 is a directory"
fi
```

Figure 13-14 Typical shell script code

The shell script is composed of different types of code. In a script file, you can use z/OS UNIX shell commands, flow control constructions like if-then-else, variables, environment settings and so on.

If you are familiar with programming languages, you will not have problems writing your first shell script.

If you are setting environment variables through a shell script, consider the following environment settings:

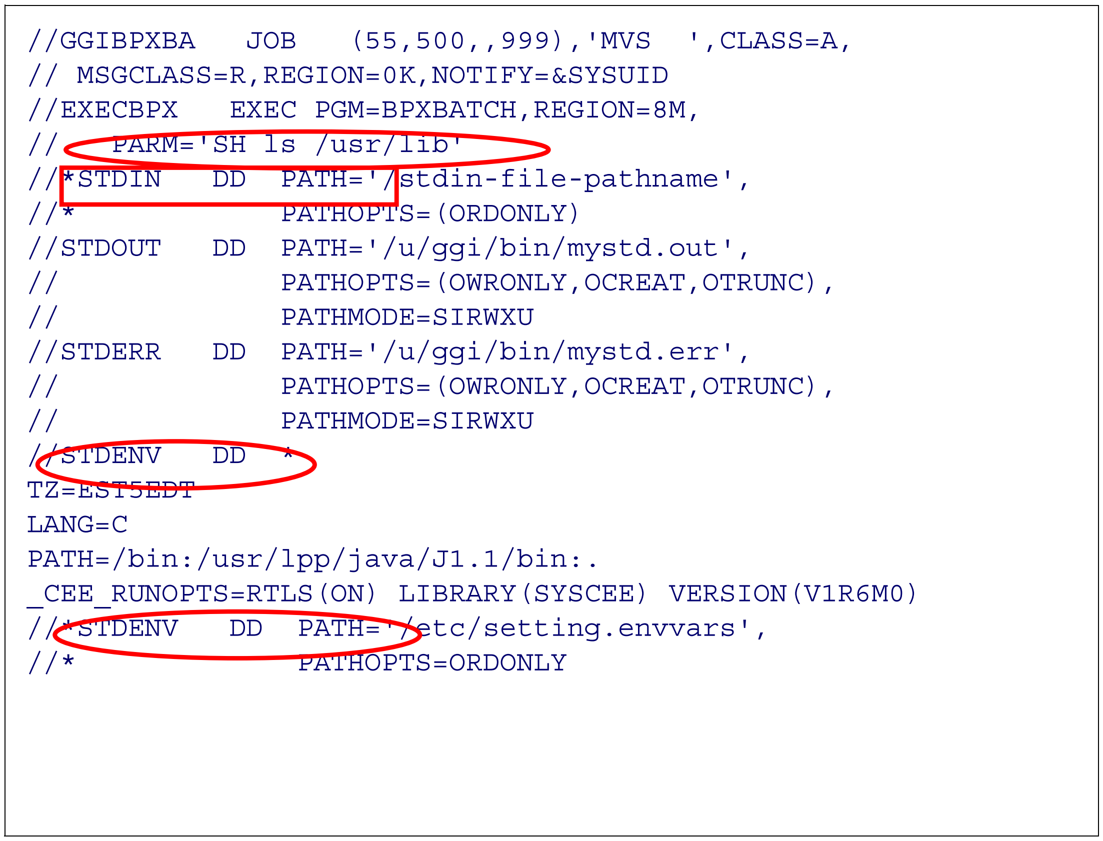
- Any variables set in a shell script are set only while the script is running and do not affect the shell that invoked the shell script (unless the script is sourced by running it with the **.** (dot) command).

To run a shell script in your current environment without creating a new process, use the **.** (dot) command. You could run the compile shell script this way:

```
. script-file-name
```

You can improve shell script performance by setting the _BPX_SPAWN_SCRIPT environment variable to a value of YES.

13.15 BPXBATCH and shell commands



```
//GGIBPXBA   JOB   (55,500,,999),'MVS  ',CLASS=A,
// MSGCLASS=R,REGION=0K,NOTIFY=&SYSUID
//EXECBPX   EXEC PGM=BPXBATCH,REGION=8M,
//     PARM='SH ls /usr/lib'
//*STDIN    DD  PATH='/stdin-file-pathname',
//*             PATHOPTS=(ORDONLY)
//STDOUT    DD  PATH='/u/ggi/bin/mystd.out',
//              PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//              PATHMODE=SIRWXU
//STDERR    DD  PATH='/u/ggi/bin/mystd.err',
//              PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//              PATHMODE=SIRWXU
//STDENV    DD  *
TZ=EST5EDT
LANG=C
PATH=/bin:/usr/lpp/java/J1.1/bin:.
_CEE_RUNOPTS=RTLS(ON) LIBRARY(SYSCEE) VERSION(V1R6M0)
//*STDENV    DD  PATH='/etc/setting.envvars',
//*              PATHOPTS=ORDONLY
```

Figure 13-15 Using shell commands in BPXBATCH JCL

BPXBATCH makes it easy for you to run shell scripts and executable files that reside in hierarchical file system (HFS) files through the MVS job control language (JCL). If you do most of your work from TSO/E, using BPXBATCH saves you the trouble of going into the shell to run your scripts and executable files. REXX execs can also use BPXBATCH to run shell scripts and executable files.

The format of BPXBATCH for JCL and TSO/E is as follows:

JCL:	EXEC PGM=BPXBATCH,PARM='SH I PGM program_name'
TSO/E:	BPXBATCH SH I PGM program_name

z/OS batch jobs that run shell commands or scripts, or z/OS C executable files in an HFS from a shell session. You can invoke BPXBATCH from a JCL job or from TSO/E (as a command, through a CALL command, or from a CLIST or REXX EXEC).

A good example is the REXX for the TSO **OSHELL** command:

```
"BPXBATCH SH "shellcmd

   IF RC ^= 0 Then
      DO
        Say ' OSHELL RC = ' RC
```

You can allocate STDIN, STDOUT, and STDERR as files, using the PATH operand, and redirect the messages to HFS files.

The command to execute the shell script or program is located in the PARM='...' section. You can either specify the PARM='...' or the //STDIN DD statements. The default when PARM='...' is not specified is SH.

SH Specifies that the shell designated in your TSO/E user ID's security product profile is to be started and is to run shell commands or scripts provided from STDIN or the specified program_name.

PGM Specifies that the specified program_name is to be run as a called program from a shell environment.

For environment settings, use the //STDENV DD statement. This can be specified as JCLIN or as a path pointing to an HFS file.

14

z/OS UNIX performance tuning

This chapter describes the tuning necessary in a z/OS UNIX environment, and also shows the required and possible settings that improve system performance.

This chapter presents the following considerations for performance tuning:

- The necessity of z/OS UNIX tuning
- The performance improvements that were achieved in the last releases of z/OS in connection to z/OS UNIX
- Settings for z/OS UNIX with Workload Manager running in compatibility mode
- Settings for z/OS UNIX with Workload Manager running in goal mode
- Additional settings to improve z/OS UNIX performance
- Where to get data to analyze the performance of z/OS UNIX

14.1 z/OS UNIX performance overview

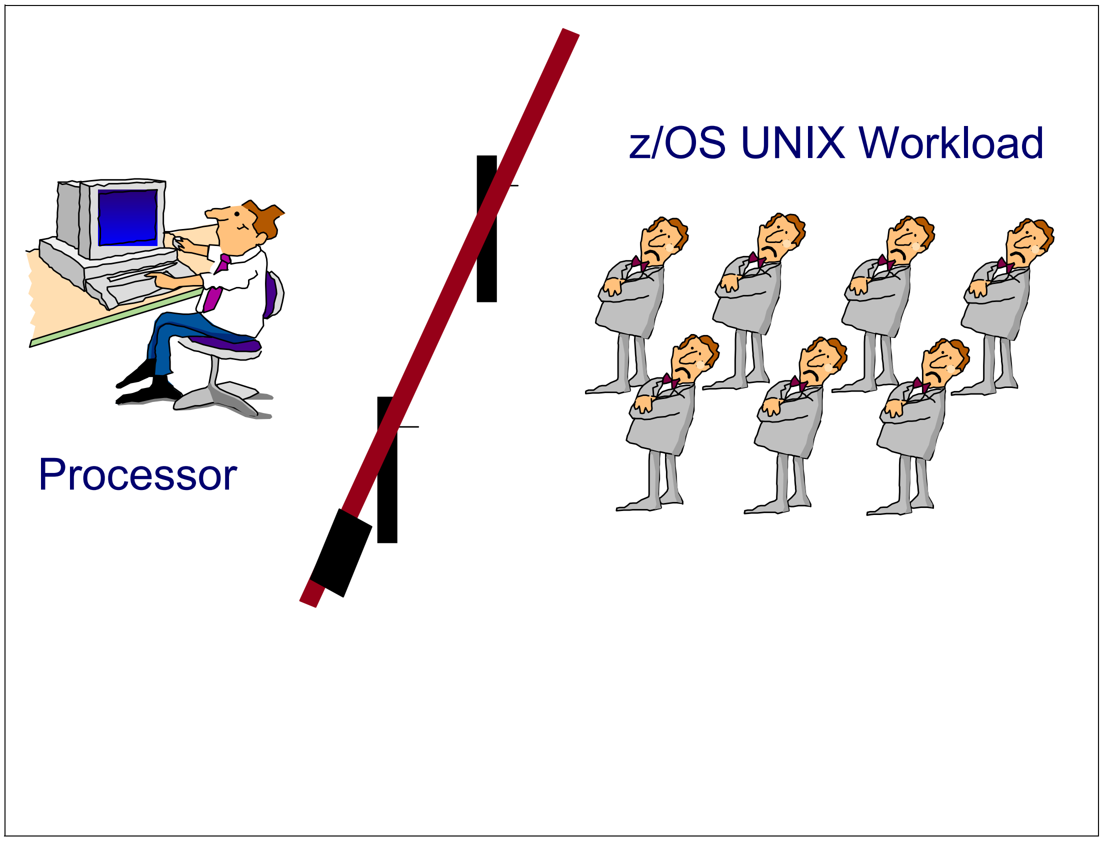


Figure 14-1 A z/OS UNIX performance overview

It is both important and necessary to tune the z/OS UNIX environment to obtain an acceptable level of performance. Because z/OS UNIX is tightly integrated into the operating system, there are many z/OS UNIX tuning steps necessary to reach maximum interoperability between z/OS UNIX and traditional z/OS services.

As you run more UNIX-based products, you need a z/OS UNIX environment that performs well. Such products are, for instance, the Lotus Domino Server, TCP/IP, SAP R/3 and the newly announced Novell Network Services based on z/OS UNIX.

Some considerations regarding your z/OS UNIX workload are the following:

- Each z/OS UNIX interactive user will consume up to double the system resource of a TSO/E user.
- Every time a user tries to invoke the z/OS UNIX shell, RACF will have to deliver the security information out of the RACF database.
- The average active user will require three or more concurrently running processes that, without tuning, run in three or more concurrently active address spaces.

These are only a few of the considerations regarding performance impacts and how to prevent them. In most cases in our lab, tuning improved throughput by 2 to 3 times and the response time improved by 2 to 5 times. That represents a significant improvement, so let‘s describe how to accomplish this.

14.2 z/OS UNIX and WLM relationship

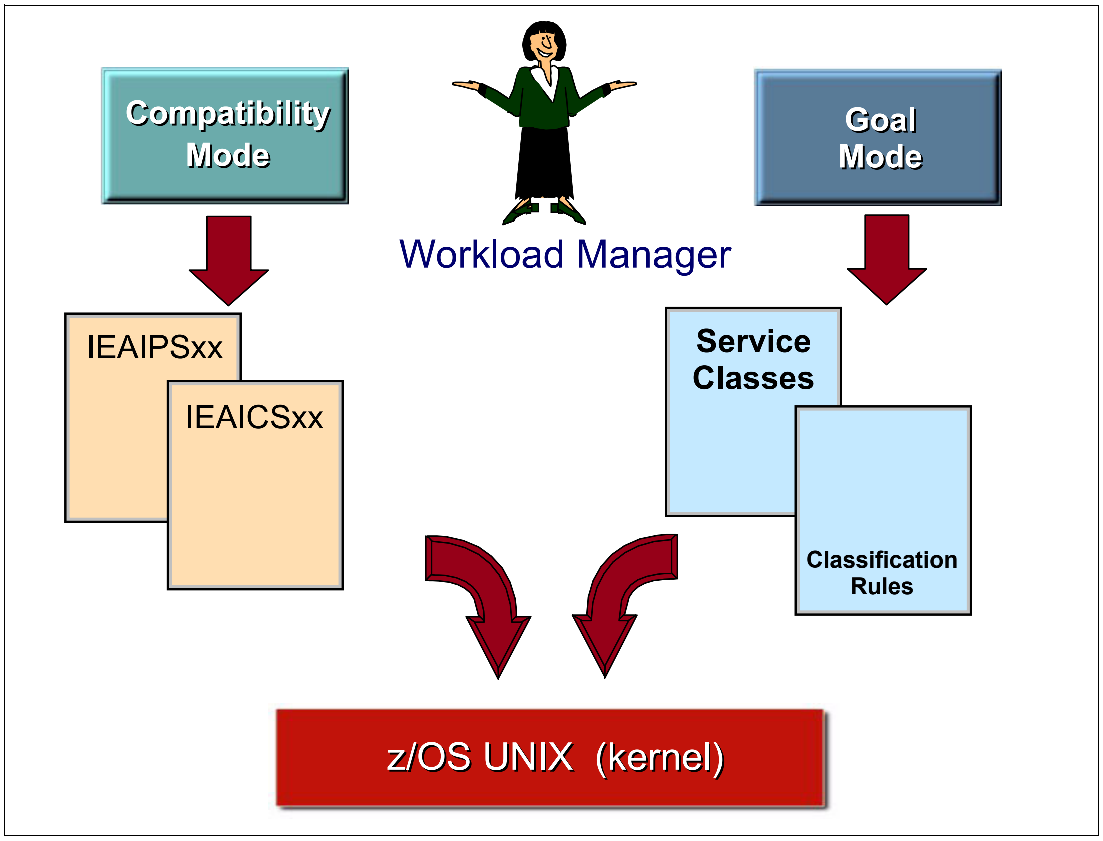


Figure 14-2 z/OS UNIX and the Workload Manager (WLM)

The Workload Manager (WLM) creates the address spaces. The BPXAS procedure found in SYS1.PROCLIB is used to create the WLM-managed address spaces. When using WLM, you do not need to issue any commands or to tune anything.

There are two modes for prioritizing kernel work in the system: the WLM compatibility mode and the WLM goal mode.

Note: Beginning with z/OS V1R3, WLM compatibility mode is no longer available.

The nice() and setpriority() kernel functions use definitions in the BPXPRMxx member of SYS1.PARMLIB for performance groups (compatibility mode) and goals (goal mode).

These definitions are optional, but if they are not specified, the nice() and setpriority() kernel functions do not change the performance level.

If there are applications that require the ability to control the priority of different processes, you must define appropriate priority levels for the application to use.

If you have enabled the `batch`, `at`, and `cron` shell functions, you need to define priority groups or goals that are appropriate for running batch jobs, as in a UNIX system.

14.3 WLM in compatibility mode

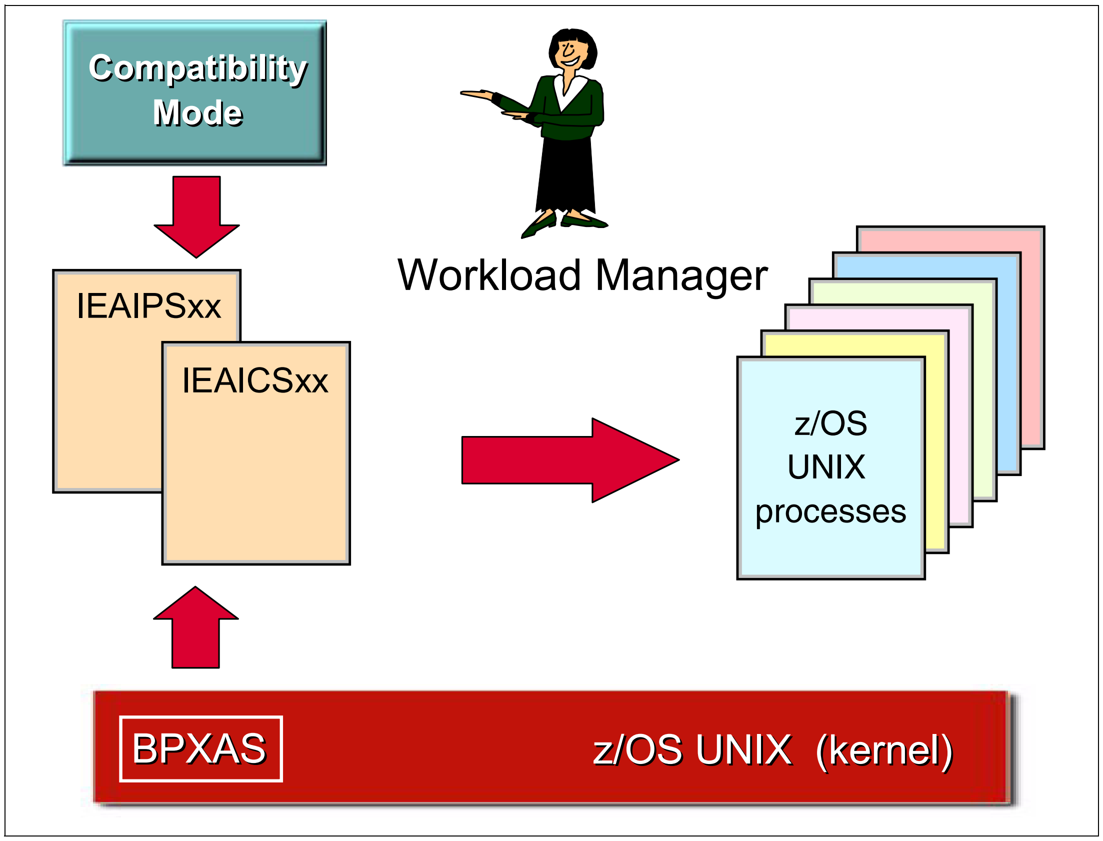


Figure 14-3 Using WLM in compatibility mode

Installations that run in WLM compatibility mode should set up performance groups for kernel services by customizing the IEAIPSxx and IEAICSxx parmlib member. If performance groups were not set up, the system will put all forked or spawned processes in the system default performance group, and this may result in wait conditions at startup time.

The IEAICSxx parmlib member has to be updated with an additional subsystem section used for z/OS UNIX specification.

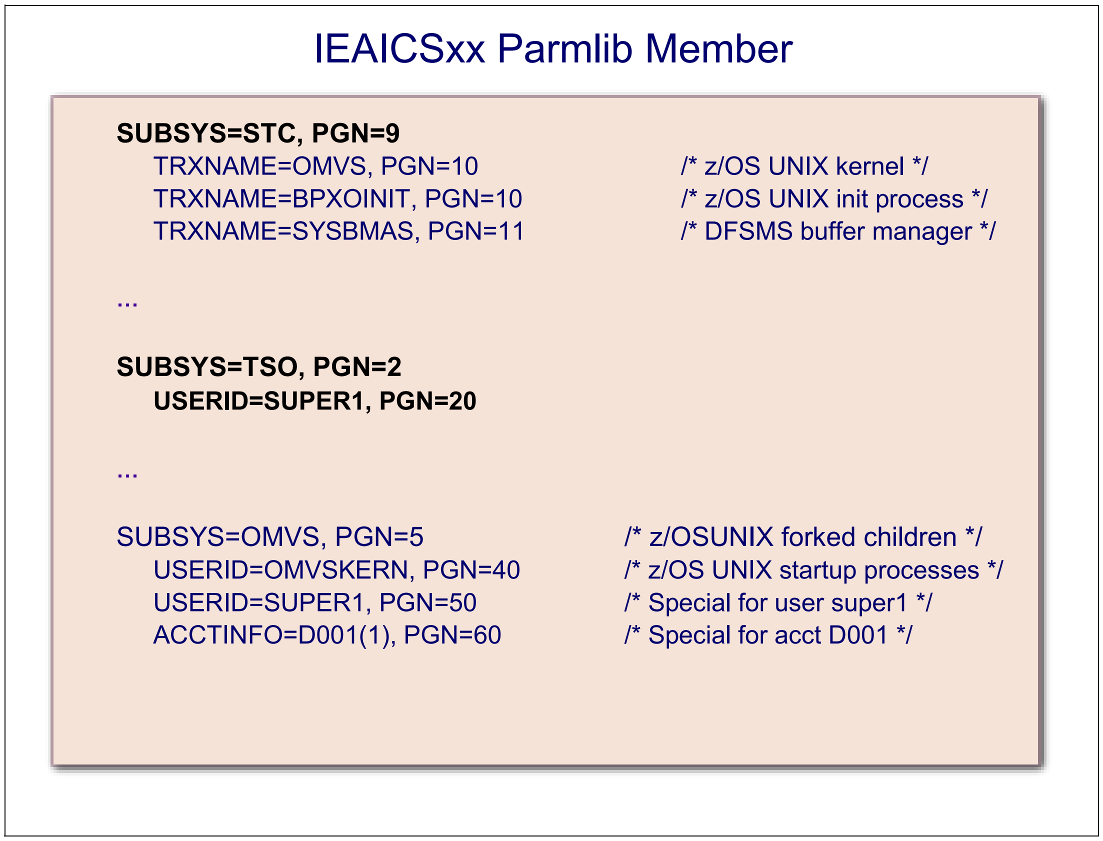
The IEAIPSxx parmlib member has to be updated with performance attributes for the performance groups added in the IEAICSxx parmlib member.

A SUBSYS=OMVS section has to be added for processing. This section is used to specify a performance group for forked address spaces. The section should specify:

- A USERID=OMVSKERN statement to specify a performance group for the startup process, which are forked by the kernel or by the initialization process BPXOINIT
- USERID, ACCTINFO or TRXNAME statements to assign different performance groups to different kernel work, as desired

The IEAIPSxx parmlib member should be updated to specify performance attributes for the performance groups added to IEAICSxx.

14.4 Customizing the IEAICSxx member


IEAICSxx Parmlib Member

```
SUBSYS=STC, PGN=9
   TRXNAME=OMVS, PGN=10                   /* z/OS UNIX kernel */
   TRXNAME=BPXOINIT, PGN=10               /* z/OS UNIX init process */
   TRXNAME=SYSBMAS, PGN=11                /* DFSMS buffer manager */

...

SUBSYS=TSO, PGN=2
   USERID=SUPER1, PGN=20

...

SUBSYS=OMVS, PGN=5                     /* z/OSUNIX forked children */
   USERID=OMVSKERN, PGN=40             /* z/OS UNIX startup processes */
   USERID=SUPER1, PGN=50               /* Special for user super1 */
   ACCTINFO=D001(1), PGN=60            /* Special for acct D001 */
```

Figure 14-4 Customizing the IEAICSxx parmlib member for z/OS UNIX

The IEAICSxx parmlib member specifies installation control for the workload in the system. Figure 14-4 shows a simplified IEAICSxx parmlib member with the statements needed for kernel services. The statements related to the kernel are highlighted.

In the SUBSYS=STC section:

- **TRXNAME=OMVS** specifies a performance group for the kernel address space. The kernel is nonswappable, but its performance group and dispatching priority are defined by the IEAICSxx and IEAIPSxx parmlib members.
- **TRXNAME=BPXOINIT** specifies a performance group for the z/OS UNIX initialization process.
- **TRXNAME=SYSBMAS** specifies the performance group for the DFSMS buffer manager used for HFS I/O.

In the SUBSYS=OMVS section, use USERID, ACCTINFO and TRXNAME statements:

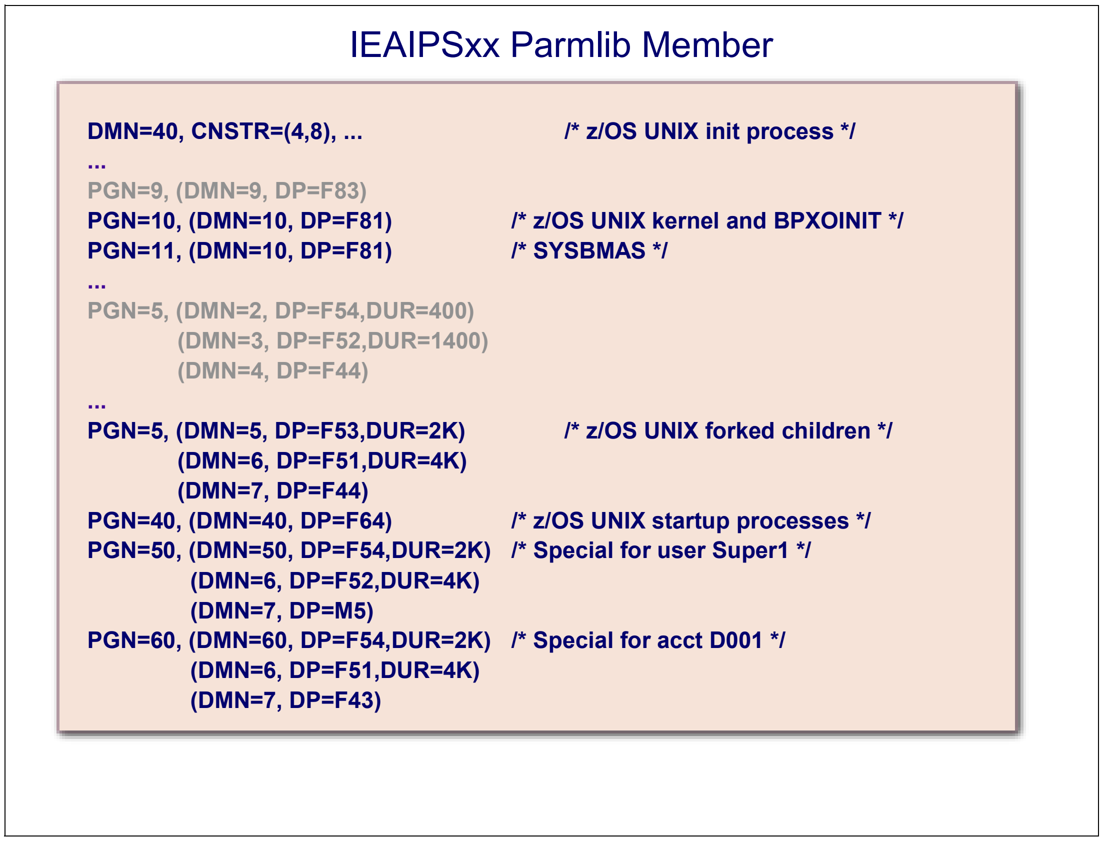
- **USERID** statements can be used to provide different users with different performance groups. Specify USERID=OMVSKERN to provide a performance group for startup processes forked by the kernel or by BPXOINIT.

  A forked child address space inherits the user ID of its parent. It may still be in a different performance group. Using the settings above, if a TSO/E user SUPER1 with PGN=20 issues a fork, the forked address space has PGN=50 from the USERID=SUPER1 statement out of the SUBSYS=OMVS section.

- **ACCTINFO** statements are necessary because fork processing always propagates accounting data from the parent to the child. Using the settings identified previously, the forked child process has been placed in performance group 60. The ACCT section begins with D001.
- **TRXNAME** statements are usable to isolate z/OS UNIX work by jobname. The default jobname value for forked or spawned processes is the user ID with a number (1-9) appended.

Note: A TRXNAME statement is the jobname for the OMVS address space. By default, fork and spawn set jobname values to the user ID with a number (1-9) appended. However, daemons or users with appropriate privileges can set the _BPX_JOBNAME environment variable to change the jobname for forked or spawned children. This way, servers and daemons in the z/OS UNIX address spaces can easily be assigned to different performance attributes other than z/OS UNIX address spaces.

14.5 Customizing the IEAIPSxx member



IEAIPSxx Parmlib Member

```
DMN=40, CNSTR=(4,8), ...                   /* z/OS UNIX init process */
...
PGN=9, (DMN=9, DP=F83)
PGN=10, (DMN=10, DP=F81)              /* z/OS UNIX kernel and BPXOINIT */
PGN=11, (DMN=10, DP=F81)              /* SYSBMAS */
...
PGN=5, (DMN=2, DP=F54,DUR=400)
        (DMN=3, DP=F52,DUR=1400)
        (DMN=4, DP=F44)
...
PGN=5, (DMN=5, DP=F53,DUR=2K)              /* z/OS UNIX forked children */
        (DMN=6, DP=F51,DUR=4K)
        (DMN=7, DP=F44)
PGN=40, (DMN=40, DP=F64)              /* z/OS UNIX startup processes */
PGN=50, (DMN=50, DP=F54,DUR=2K)  /* Special for user Super1 */
         (DMN=6, DP=F52,DUR=4K)
         (DMN=7, DP=M5)
PGN=60, (DMN=60, DP=F54,DUR=2K)  /* Special for acct D001 */
         (DMN=6, DP=F51,DUR=4K)
         (DMN=7, DP=F43)
```

Figure 14-5 Customizing the IEAIPSxx member for z/OS UNIX

The IEAIPSxx parmlib member provides installation performance specifications for the performance groups in the IEAICSxx parmlib member. The example in Figure 14-5 contains performance specifications based on the performance group numbers specified in the IEAICSxx member.

Use the following rules when customizing the IEAIPSxx parmlib member statements:

- Give the kernel and the z/OS UNIX initialization process, BPXOINIT, high priority.
- The priority of started task SUBSYS=STC, TRXNAME=SYSBMAS, which is the DFSMS buffer manager, must be higher than the priority of any user accessing HFS files.
- Allow the startup processes a multiprogramming level (MPL) high enough to support the /usr/sbin/init process and any forked child processes (daemons).
- The MPL value for started tasks must be high enough to include all forked initiators provided by WLM.
- Provide normal forked children with more than one performance period. z/OS UNIX work can range from quick built-in shell commands to long-running background processes. The service requirements for some of these address spaces may be similar to medium-length TSO/E transactions, and for others, similar to long-running batch jobs.
- If you have used the PRIORITYPG statement in the BPXPRMxx parmlib member to enable nice(), setpriority(), and chpriority(), additional performance groups for z/OS UNIX work must be added.

14.6 WLM in goal mode

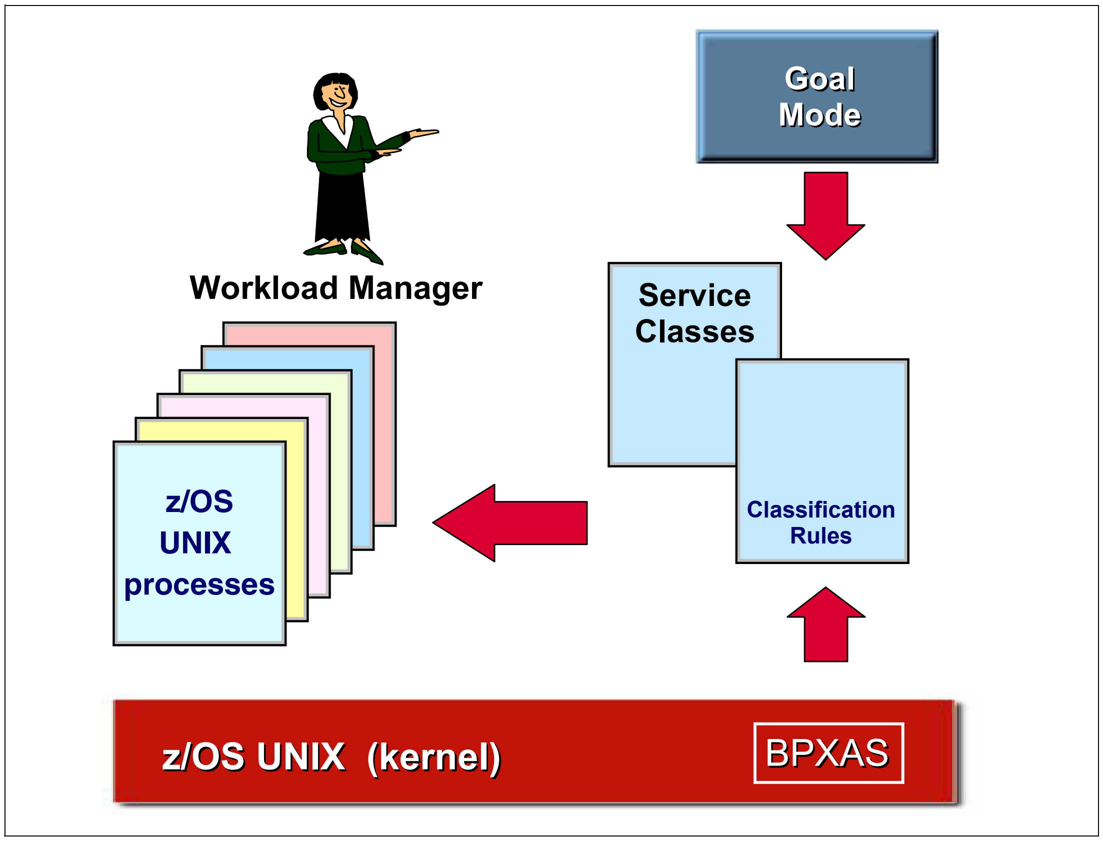


Figure 14-6 Using WLM in goal mode

Installations that run in goal mode can take the following steps to customize service policies in their Workload Manager service definition:

1. Define a workload for z/OS UNIX kernel work.
2. Define service classes for z/OS UNIX kernel work:
    a. Define a service class for forked children.
    b. Define a service class for startup processes.
3. Define classification rules:
    a. By default, put child processes (subsystem type OMVS) into the service class defined for forked children.
    b. Put the kernel (TRXNAME=OMVS) into a high-priority started task (subsystem type STC) service class.
    c. Put the initialization process BPXOINIT (TRXNAME=BPXOINIT) into a high-priority started task (subsystem type STC) service class.
    d. Startup processes that are forked by the initialization process, BPXOINIT, fall under SUBSYS=OMVS.
    e. Other forked child processes (under subsystem type OMVS) can be assigned to different service classes.
    f. Put the DFSMS buffer manager SYSBMAS (TRXNAME=SYSBMAS) into a high-priority started task (subsystem type STC) service class.

14.7 Defining service classes

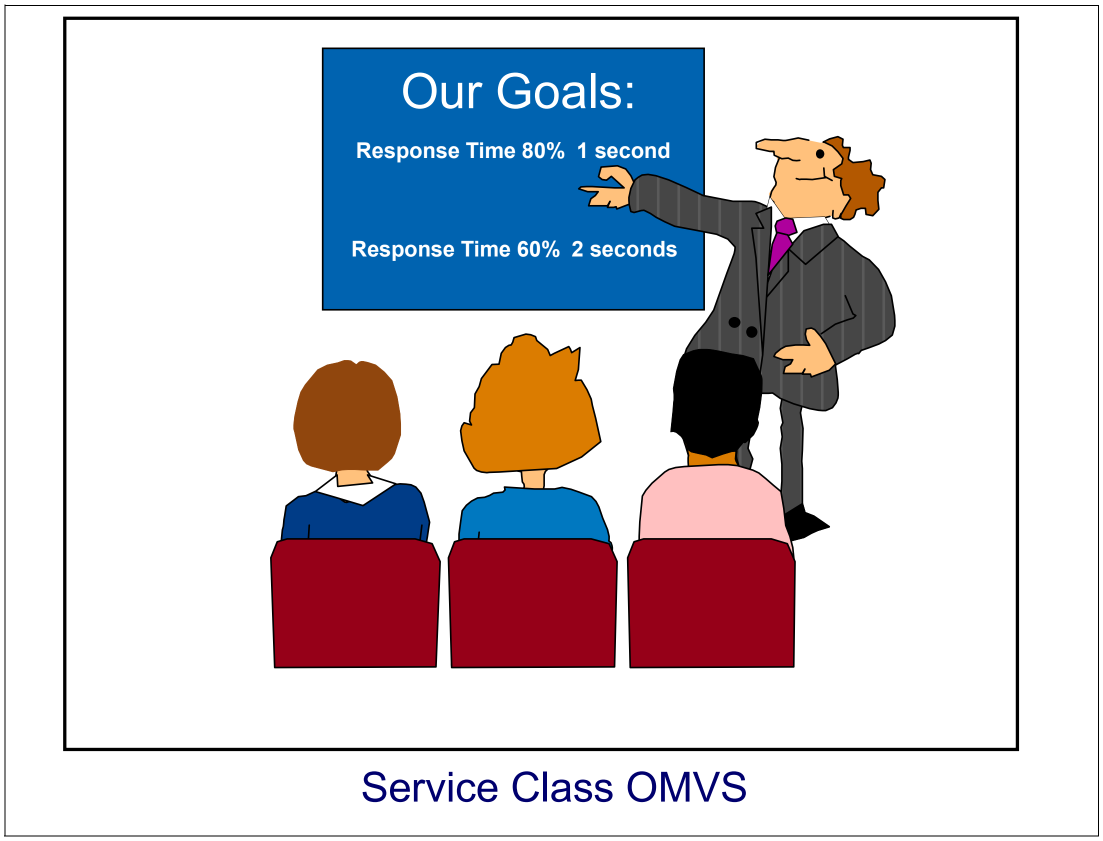


Figure 14-7 Defining service classes for OMVS address spaces

Define a service class for forked child address spaces. This service class should normally have three performance periods, because it must support all types of kernel work, from short interactive commands to long-running background work. The following is a sample service class for forked children. You should change the values as appropriate for the corresponding installations.

Table 14-1 Service Class OMVS: Base goals

#	Duration	Importance	Goal Description
1	2000	2	Response Time 80% 1 second
2	4000	3	Response Time 60% 2 seconds
3		5	Execution Velocity of 10

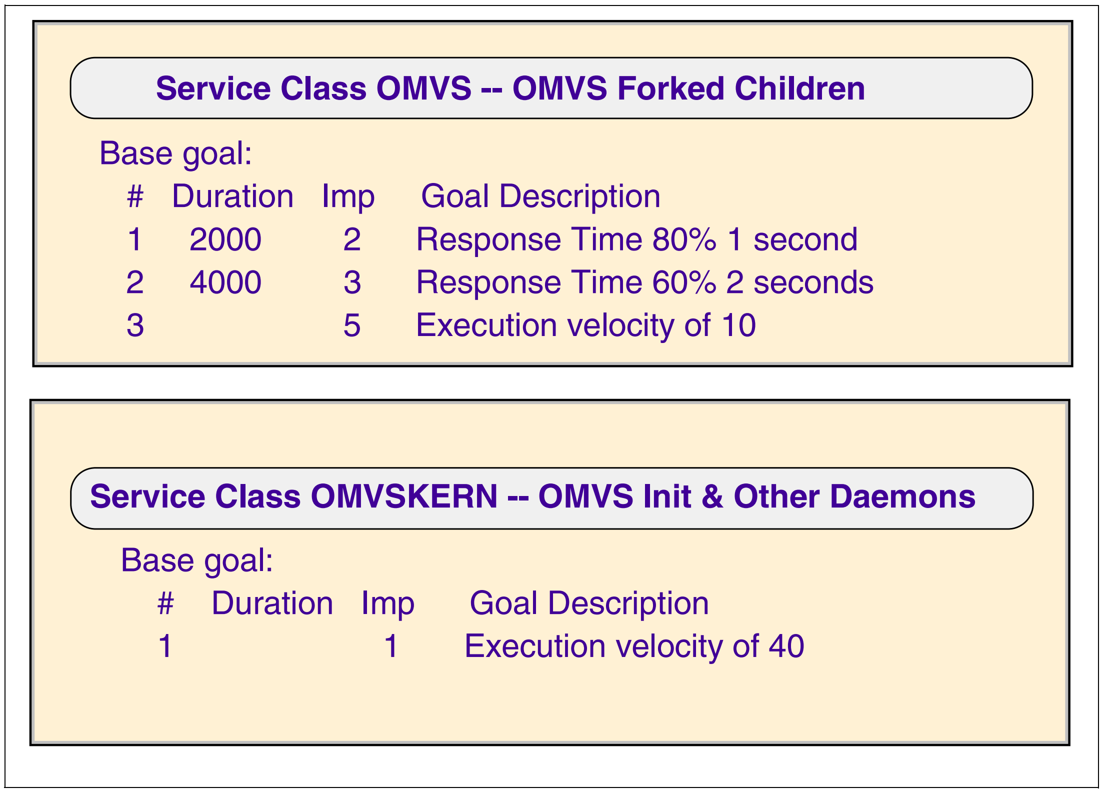
Also define a service class for daemons. This service class should normally have only one period with a velocity goal higher than the velocity goals of other forked children.

Table 14-2 Service Class OMVSKERN: Base goals

#	Duration	Importance	Goal Description
1		1	Execution Velocity of 40

You may want to define other service classes for z/OS UNIX kernel work for special users.

14.8 Workload Manager service classes



Service Class OMVS -- OMVS Forked Children

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

Service Class OMVSKERN -- OMVS Init & Other Daemons

Base goal:

#	Duration	Imp	Goal Description
1		1	Execution velocity of 40

Figure 14-8 Service class definitions for OMVS processes

When using Workload Manager in goal mode, the z/OS workloads must be defined to Workload Manager.

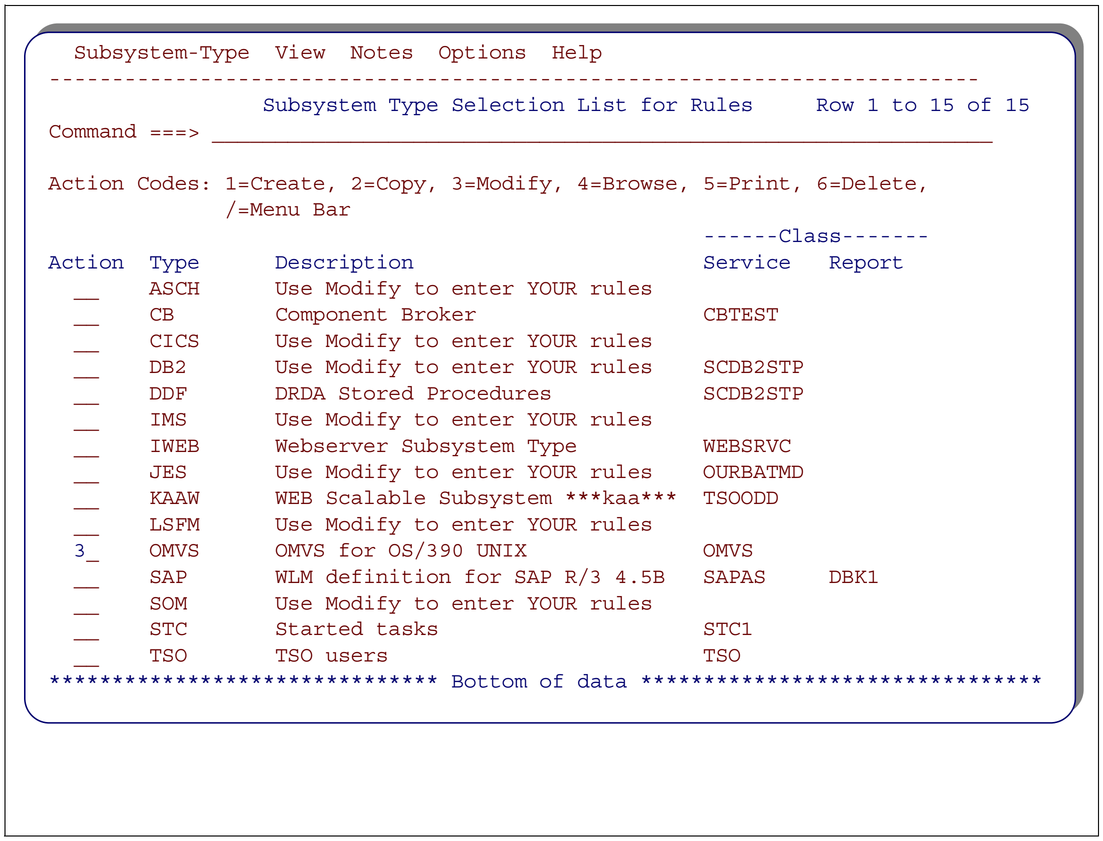
Define a service class called OMVS which will include all the forked or spawned child address spaces. You should define three performance periods for this class because it must support all types of z/OS work; from short interactive commands to long running background work.

Define a separate service class for the OMVSKERN userid which will include the initialization process and other daemons started at initialization time by forks from this process. This service class should have only one period with a velocity goal higher than the goals for the forked children (OMVS service class).

The service classes which you define to Workload Manager are similar to the definitions in the IEAIPS member. Performance goals for a service class can be defined as either response time, velocity, or discretionary. You can use either response goals or velocity for z/OS UNIX work.

An installation may choose to have other service classes defined for special z/OS UNIX users in addition to the one for OMVSKERN. A good example might be daemons such as INETD or TELNETD. These daemons run under the OMVSKERN ID by default, but it is possible to assign different RACF user IDs to the daemons, which can then be separately controlled.

14.9 Subsystem type panel



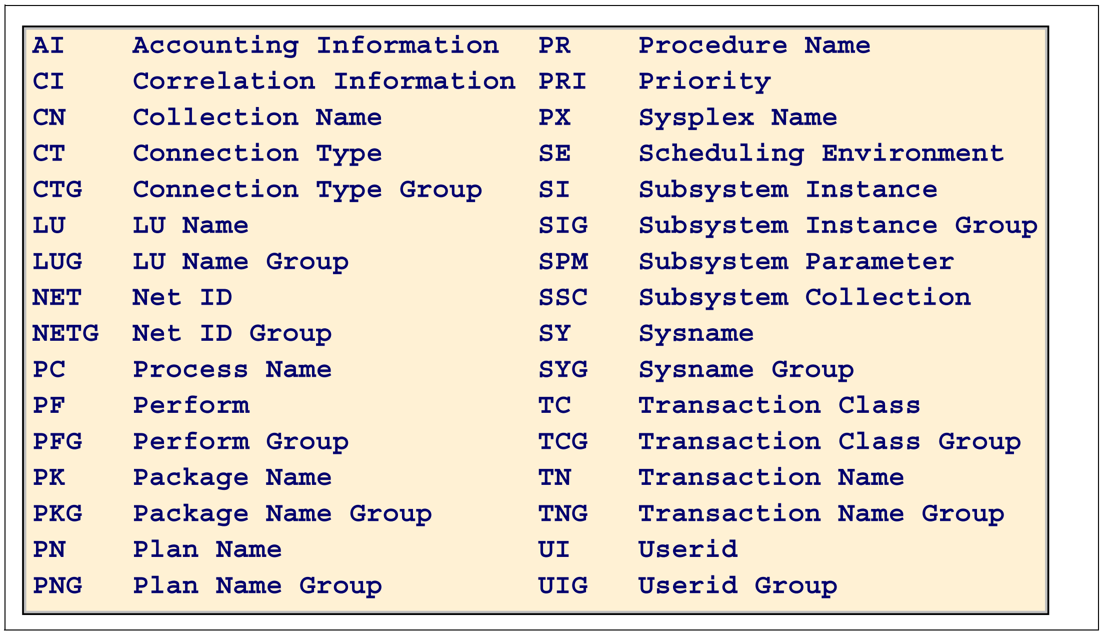
```
  Subsystem-Type  View  Notes  Options  Help
--------------------------------------------------------------------------
                 Subsystem Type Selection List for Rules     Row 1 to 15 of 15
Command ===> ______________________________________________________________

Action Codes: 1=Create, 2=Copy, 3=Modify, 4=Browse, 5=Print, 6=Delete,
              /=Menu Bar
                                                    ------Class-------
Action  Type      Description                       Service   Report
  __    ASCH      Use Modify to enter YOUR rules
  __    CB        Component Broker                  CBTEST
  __    CICS      Use Modify to enter YOUR rules
  __    DB2       Use Modify to enter YOUR rules    SCDB2STP
  __    DDF       DRDA Stored Procedures            SCDB2STP
  __    IMS       Use Modify to enter YOUR rules
  __    IWEB      Webserver Subsystem Type          WEBSRVC
  __    JES       Use Modify to enter YOUR rules    OURBATMD
  __    KAAW      WEB Scalable Subsystem ***kaa***  TSOODD
  __    LSFM      Use Modify to enter YOUR rules
  3_    OMVS      OMVS for OS/390 UNIX              OMVS
  __    SAP       WLM definition for SAP R/3 4.5B   SAPAS     DBK1
  __    SOM       Use Modify to enter YOUR rules
  __    STC       Started tasks                     STC1
  __    TSO       TSO users                         TSO
******************************* Bottom of data ********************************
```

Figure 14-9 Selecting the OMVS subsystem to create classification rules

Figure 14-9 shows the IBM-supplied subsystem types that workload management supports.

The ISPF application provides these subsystem types as a selection list on the classification rules panel. You can add any additional subsystem type if it supports workload management on the same panel.

14.10 WLM work qualifiers



AI	Accounting Information	PR	Procedure Name
CI	Correlation Information	PRI	Priority
CN	Collection Name	PX	Sysplex Name
CT	Connection Type	SE	Scheduling Environment
CTG	Connection Type Group	SI	Subsystem Instance
LU	LU Name	SIG	Subsystem Instance Group
LUG	LU Name Group	SPM	Subsystem Parameter
NET	Net ID	SSC	Subsystem Collection
NETG	Net ID Group	SY	Sysname
PC	Process Name	SYG	Sysname Group
PF	Perform	TC	Transaction Class
PFG	Perform Group	TCG	Transaction Class Group
PK	Package Name	TN	Transaction Name
PKG	Package Name Group	TNG	Transaction Name Group
PN	Plan Name	UI	Userid
PNG	Plan Name Group	UIG	Userid Group

Figure 14-10 The WLM work qualifiers used to classify work

A work qualifier is an attribute of incoming work. Figure 14-10 shows a list of the work qualifiers that may be used in defining classification rules that determine the service class a unit of work is assigned.

Each subsystem has its own list of work qualifiers that it supports.

14.11 OMVS work qualifiers

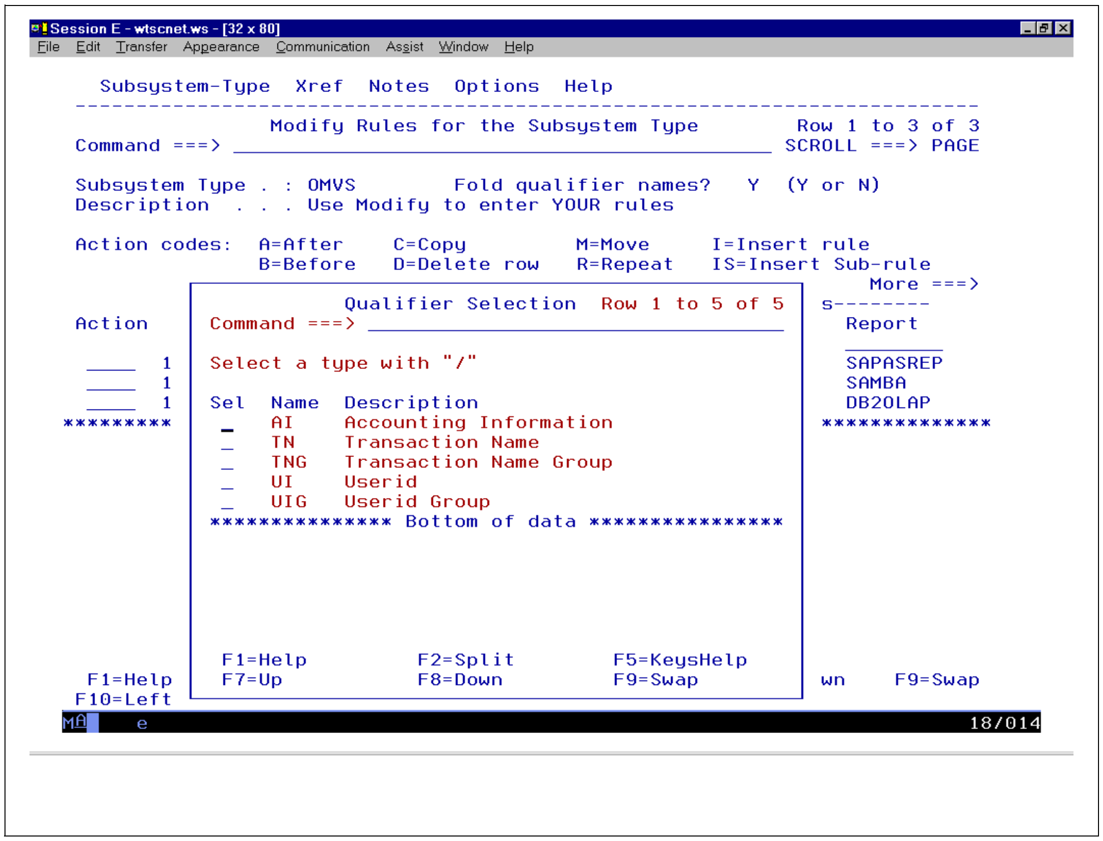
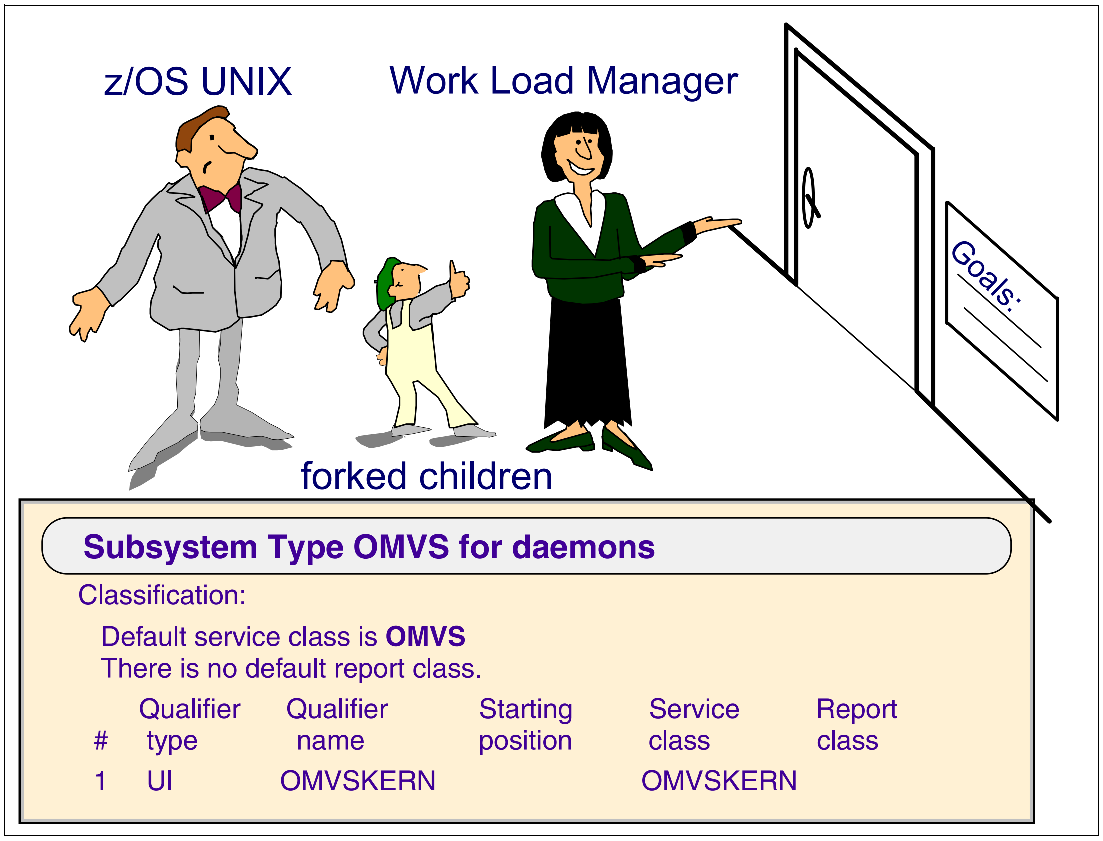

```
Session E - wtscnet.ws - [32 x 80]
File  Edit  Transfer  Appearance  Communication  Assist  Window  Help

  Subsystem-Type  Xref  Notes  Options  Help
 --------------------------------------------------------------------------
                Modify Rules for the Subsystem Type         Row 1 to 3 of 3
Command ===> ____________________________________________  SCROLL ===> PAGE

Subsystem Type . : OMVS         Fold qualifier names?   Y  (Y or N)
Description  . . . Use Modify to enter YOUR rules

Action codes:  A=After    C=Copy         M=Move     I=Insert rule
               B=Before   D=Delete row   R=Repeat   IS=Insert Sub-rule
                                                          More ===>
           |          Qualifier Selection  Row 1 to 5 of 5 |  s--------
Action     | Command ===> _________________________________ |    Report
           |                                                |
 ____   1  | Select a type with "/"                         |    ________
 ____   1  |                                                |    SAPASREP
 ____   1  | Sel  Name  Description                         |    SAMBA
*********  |  _   AI    Accounting Information              |    DB2OLAP
           |  _   TN    Transaction Name                    |  **************
           |  _   TNG   Transaction Name Group              |
           |  _   UI    Userid                              |
           |  _   UIG   Userid Group                        |
           | *************** Bottom of data **************** |
           |                                                |
           |                                                |
           |   F1=Help         F2=Split        F5=KeysHelp  |
 F1=Help   |   F7=Up           F8=Down         F9=Swap      |  wn     F9=Swap
F10=Left   |________________________________________________|
MA     e                                                              18/014
```

Figure 14-11 The work qualifiers usable for OMVS

Accounting data is normally inherited from the parent process of a z/OS UNIX System Services address space. In addition, when a daemon creates a process for another user, accounting data is taken from the WORKATTR of the RACF user profile. A user can also assign accounting data by setting the _BPX_ACCT_DATA environment variable or by passing accounting data on the interface to the _spawn service. For more information about z/OS UNIX System Services accounting information, see *z/OS UNIX System Services Planning*, GA22-7800.

Figure 14-11 shows the work qualifiers that can be used to set up the OMVS classification rules.

14.12 Defining classification rules



Subsystem Type OMVS for daemons

Classification:

Default service class is **OMVS**
There is no default report class.

#	Qualifier type	Qualifier name	Starting position	Service class	Report class
1	UI	OMVSKERN		OMVSKERN	

Figure 14-12 Defining classification rules for OMVS

The workload in a z/OS system must be identified and classified in Workload Manager. The classification rules should specify a subsystem called OMVS, which will cover all forked/spawned child processes. Within this subsystem, the userid OMVSKERN should be defined to use a separate service class than the rest of the work in this subsystem. Other categories can be defined if you have separated daemons out into other service classes. Specify the classification rules needed to separate daemons (for example, inetd) from other forked children. The following is a sample classification for subsystem type OMVS.

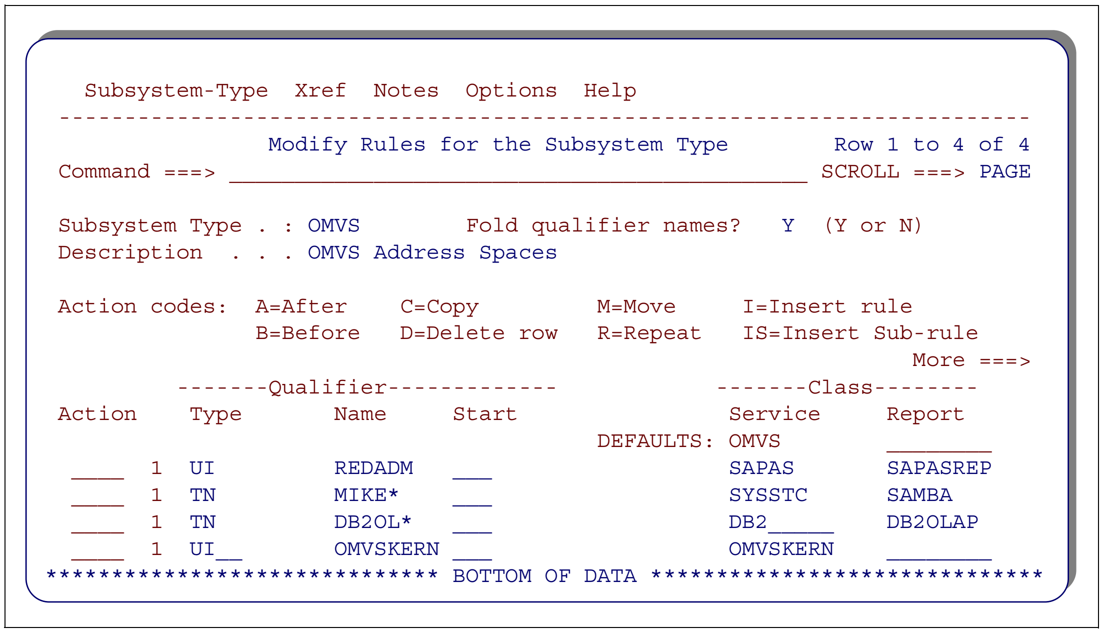
- Subsystem Type OMVS
  - Classification:
    - Default service class is OMVS.
    - There is no default report class.

Table 14-3 Classification Rules for Subsystem Type OMVS

#	Qualifier Type	Qualifier Name	Starting Position	Service Class	Report Class
1	UI	OMVSKERN		OMVSKERN	

If no action was taken, OMVS and BPXOINIT will run under the rules for subsystem type STC, which is typically defined to have high priority. If needed, it is possible to define an extra classification rule for subsystem type STC to ensure that the kernel, the initialization process BPXOINIT, and the DFSMS buffer manager SYSBMAS run as high-priority started tasks.

14.13 Classification rules



```
  Subsystem-Type  Xref  Notes  Options  Help
--------------------------------------------------------------------------
                Modify Rules for the Subsystem Type        Row 1 to 4 of 4
Command ===> ____________________________________________ SCROLL ===> PAGE

Subsystem Type . : OMVS        Fold qualifier names?   Y  (Y or N)
Description  . . . OMVS Address Spaces

Action codes:  A=After    C=Copy        M=Move     I=Insert rule
               B=Before   D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                              More ===>
         -------Qualifier-------------             -------Class--------
Action     Type        Name      Start                 Service     Report
                                           DEFAULTS: OMVS        ________
 ____     1  UI         REDADM    ___                   SAPAS       SAPASREP
 ____     1  TN         MIKE*     ___                   SYSSTC      SAMBA
 ____     1  TN         DB2OL*    ___                   DB2_____    DB2OLAP
 ____     1  UI__       OMVSKERN  ___                   OMVSKERN    ________
****************************** BOTTOM OF DATA ******************************
```

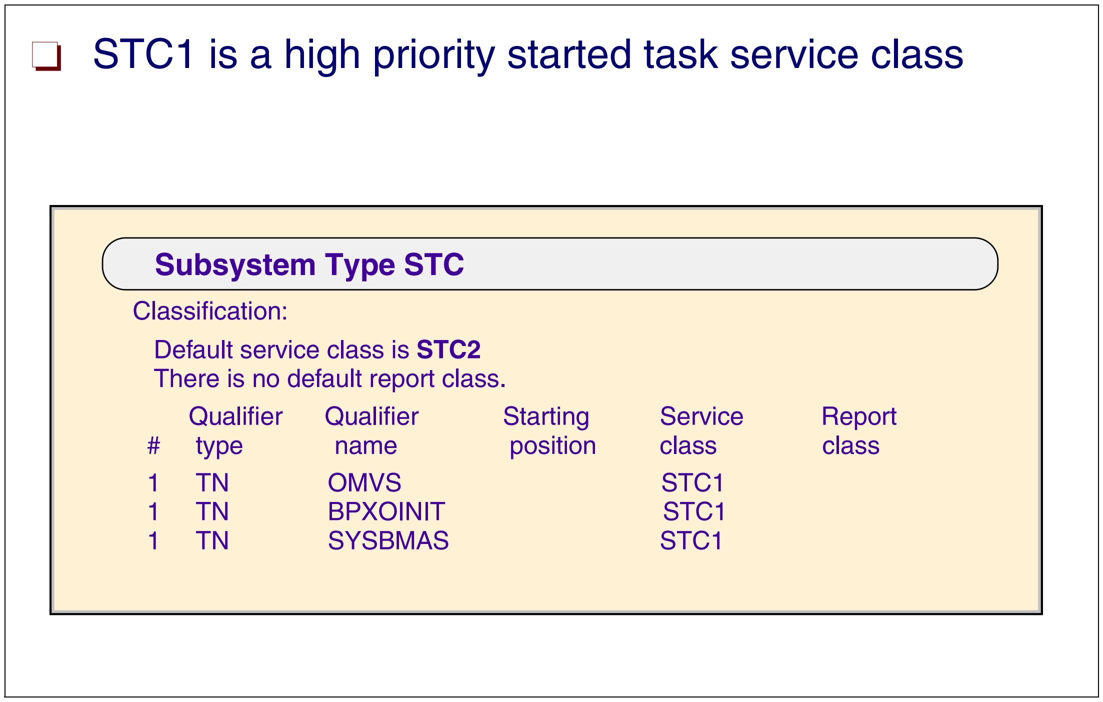
Figure 14-13 Panel to define the classification rules

When the subsystem receives a work request, the system searches the classification rules for a matching qualifier and its service class or report class. Because a piece of work can have more than one work qualifier associated with it, it may match more than one classification rule. Therefore, the order in which you specify the classification rules determines which service classes are assigned.

A service class default is the service class that is assigned if no other classification rule matches for that subsystem type. If you want to assign any work in a subsystem type to a service class, then you must assign a default service class for that subsystem -- except for STC. You are not required to assign a default service class for STC, even if you assign started tasks to different service classes.

Optionally, you can assign a default report class for the subsystem type. If you want to assign work running in a subsystem to report classes, then you do not have to assign a default service class for that subsystem.

14.14 Classification rules for STC



❑ STC1 is a high priority started task service class

Subsystem Type STC

Classification:

Default service class is **STC2**
There is no default report class.

#	Qualifier type	Qualifier name	Starting position	Service class	Report class
1	TN	OMVS		STC1	
1	TN	BPXOINIT		STC1	
1	TN	SYSBMAS		STC1	

Figure 14-14 Panel for defining the STC service classes for OMVS

Specify a classification rule for subsystem type STC to ensure that the OMVS, BPXOINIT, and SYSBMAS procedures run as high priority started tasks, as follows:

- Subsystem Type STC
  - Classification:
    - Default service class is STC2.
    - There is no default report class.

Table 14-4 Classification Rules for Subsystem Type OMVS

#	Qualifier Type	Qualifier Name	Starting Position	Service Class	Report Class
1	UI	OMVS		STC1	
2	UI	BPXOINIT		STC1	
3	UI	SYSBMAS		STC1	
..	...	....		....	

14.15 Virtual lookaside facility (VLF)

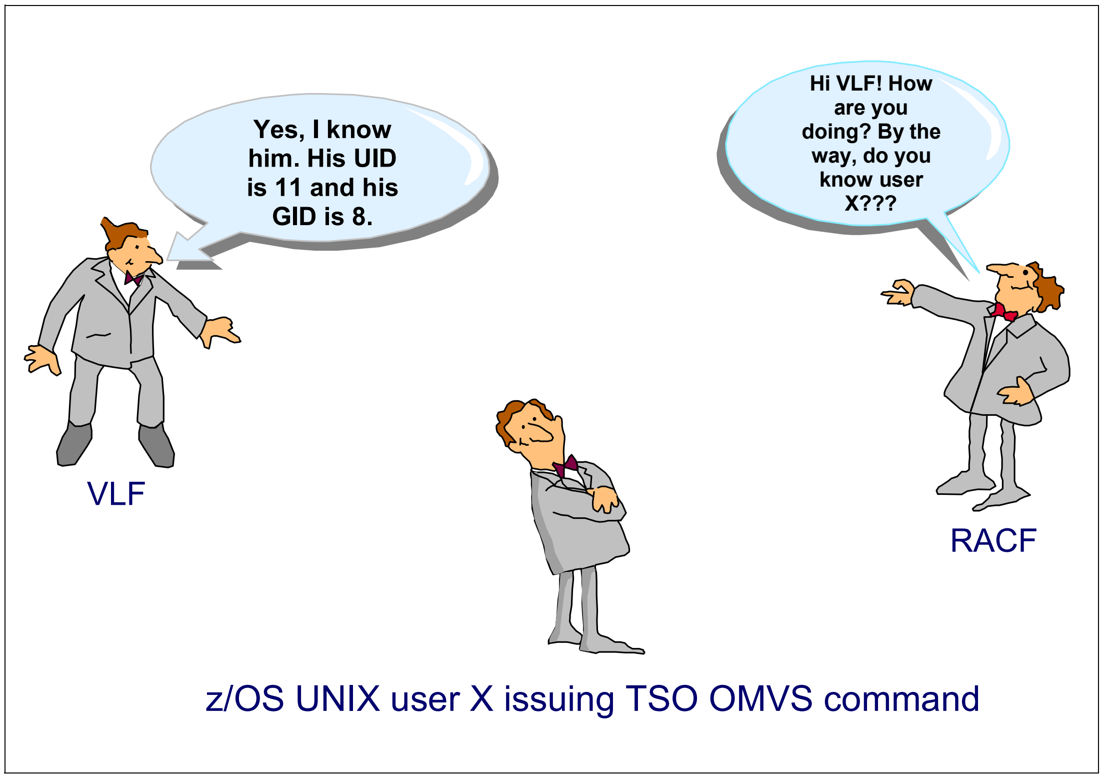

Yes, I know him. His UID is 11 and his GID is 8.
Hi VLF! How are you doing? By the way, do you know user X???
VLF
RACF
z/OS UNIX user X issuing TSO OMVS command

Figure 14-15 Using VLF to cache UIDs and GIDs

To improve the performance of z/OS UNIX RACF support, you should add the Virtual Lookaside Facility (VLF) classes that control caching of the z/OS UNIX UID and GID information. The mapping of z/OS UNIX group identifiers (GIDs) to RACF group names and the mapping of z/OS UNIX user identifiers (UIDs) to RACF user IDs is implemented using VLF services to gain performance improvements.

To achieve these performance improvements, the two VLF classes IRRGMAP and IRRUMAP should be added to the COFVLFxx parmlib member. A COFVLFxx parmlib member including the z/OS UNIX information should look like this:

```
CLASS NAME(CSVLLA)          /* Class name for LLA                */
   EMAJ(LLA)                /* Major name for LLA                */
CLASS NAME(IRRUMAP)         /* z/OS UNIX RACF UMAP Table       */
   EMAJ(UMAP)               /* Enable Caching of z/OS UNIX UIDs */
CLASS NAME(IRRGMAP)         /* z/OS UNIX RACF GMAP Table       */
   EMAJ(GMAP)               /* Enable Caching of z/OS UNIX GIDs */
CLASS NAME(IRRGTS)          /* RACF GTS Table                    */
   EMAJ(GTS)                /* Enable caching of RACF GTS        */
CLASS NAME(IRRACEE)         /* RACF saved ACEEs                  */
   EMAJ(ACEE)               /* Enable caching of RACF ACEE       */
CLASS NAME(IRRSMAP)         /* z/OS UNIX Security Package      */
   EMAJ(SMAP)               /* Major Node SMAP                   */
```

The VLF member can be activated by starting VLF using the operator command:

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

where xx is the suffix of the corresponding COFVLF member.

If a user requests to invoke z/OS UNIX, the z/OS UNIX kernel asks RACF about the permission. RACF checks whether VLF is active. If it is active, it asks VLF about the data of the user that tries to logon. If the user has already been logged to z/OS UNIX since VLF became active, VLF should know about this user and provides the UID and GID to RACF. RACF passes the information to z/OS UNIX for processing.

However, if VLF is not active, or if the user tries to invoke z/OS UNIX for the first time since VLF became active, RACF has to start I/O to the RACF database to get the information.

VLF is able to collect this data in its data spaces if the following two classes were added to the COFVLFxx member in SYS1.PARMLIB:

- IRRUMAP
- IRRGMAP

14.16 VLF for z/OS UNIX

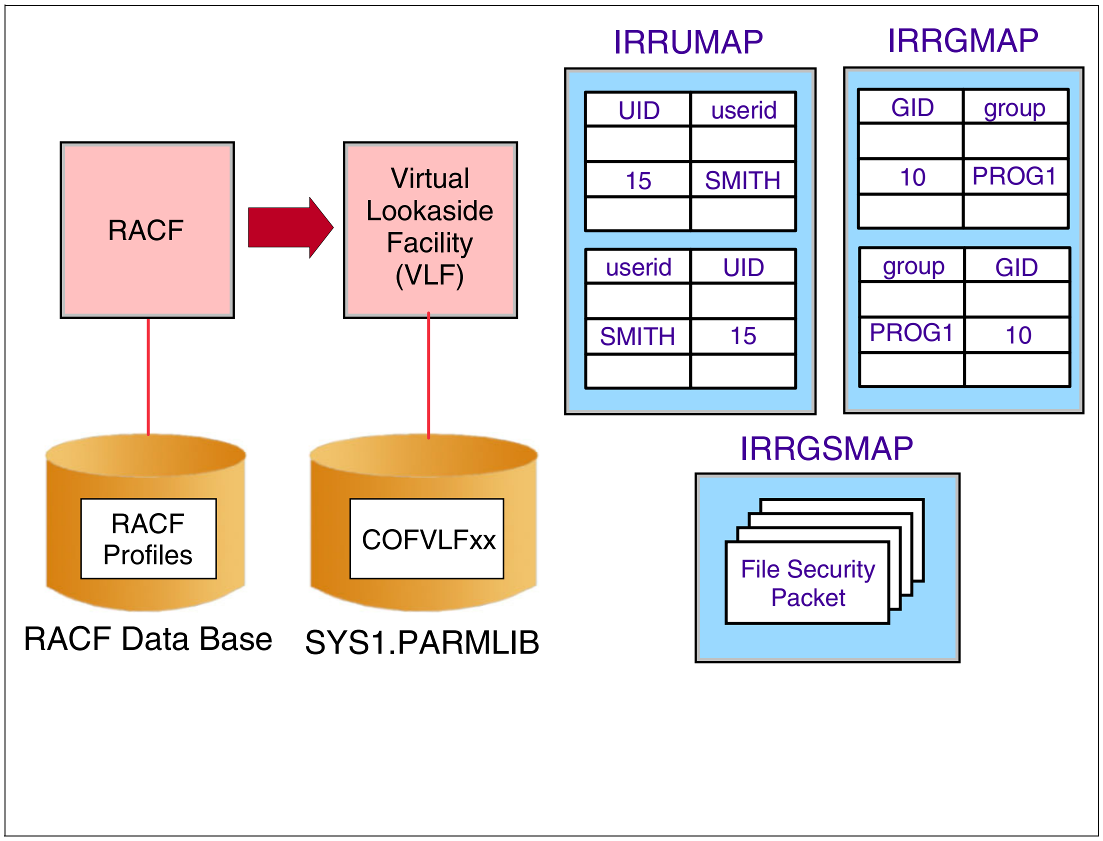


Figure 14-16 The VLF cache of UIDs and GIDs with RACF

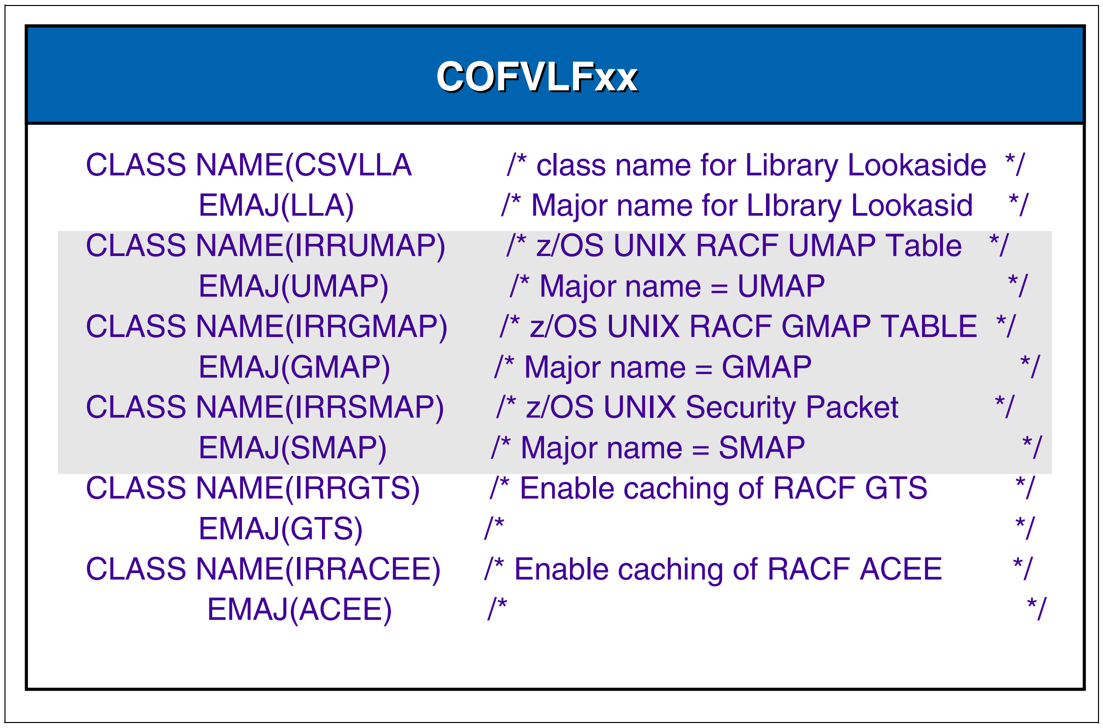
z/OS UNIX uses the UID and the GID to identify users and groups while z/OS uses the userid and group name. Both identifications are defined in the RACF profiles in the RACF database. When RACF is called to do security processing for z/OS UNIX, RACF often needs to find which userid belongs to a UID, or which group belongs to a GID. This could cause excessive I/O to the RACF database. A solution is to build tables with UID-to-userid mapping and GID-to-group mapping and keep the tables in VLF dataspaces called IRRUMAP and IRRGMAP.

Also, whenever a file is accessed by a user, RACF must check the permission bits, which are stored in a File Security Packet (FSP) on the same HFS data set as the file. As a user may access the same file many times, RACF can save time and I/O by storing File Security Packets for active files in a VLF dataspace called IRRSMAP.

Virtual Lookaside Facility (VLF) is used to build the tables and keep them in processor storage. For this mechanism to work:

- VLF must be active.
- The IRRUMAP and IRRGMAP classes must be defined.

14.17 COFVLFxx updates for z/OS UNIX



COFVLFxx

```
CLASS NAME(CSVLLA          /* class name for Library Lookaside  */
        EMAJ(LLA)          /* Major name for LIbrary Lookasid   */
CLASS NAME(IRRUMAP)        /* z/OS UNIX RACF UMAP Table   */
        EMAJ(UMAP)         /* Major name = UMAP                 */
CLASS NAME(IRRGMAP)        /* z/OS UNIX RACF GMAP TABLE  */
        EMAJ(GMAP)         /* Major name = GMAP                 */
CLASS NAME(IRRSMAP)        /* z/OS UNIX Security Packet         */
        EMAJ(SMAP)         /* Major name = SMAP                 */
CLASS NAME(IRRGTS)         /* Enable caching of RACF GTS        */
        EMAJ(GTS)          /*                                   */
CLASS NAME(IRRACEE)        /* Enable caching of RACF ACEE       */
        EMAJ(ACEE)         /*                                   */
```

Figure 14-17 The definitions required in the COFVLFxx parmlib member

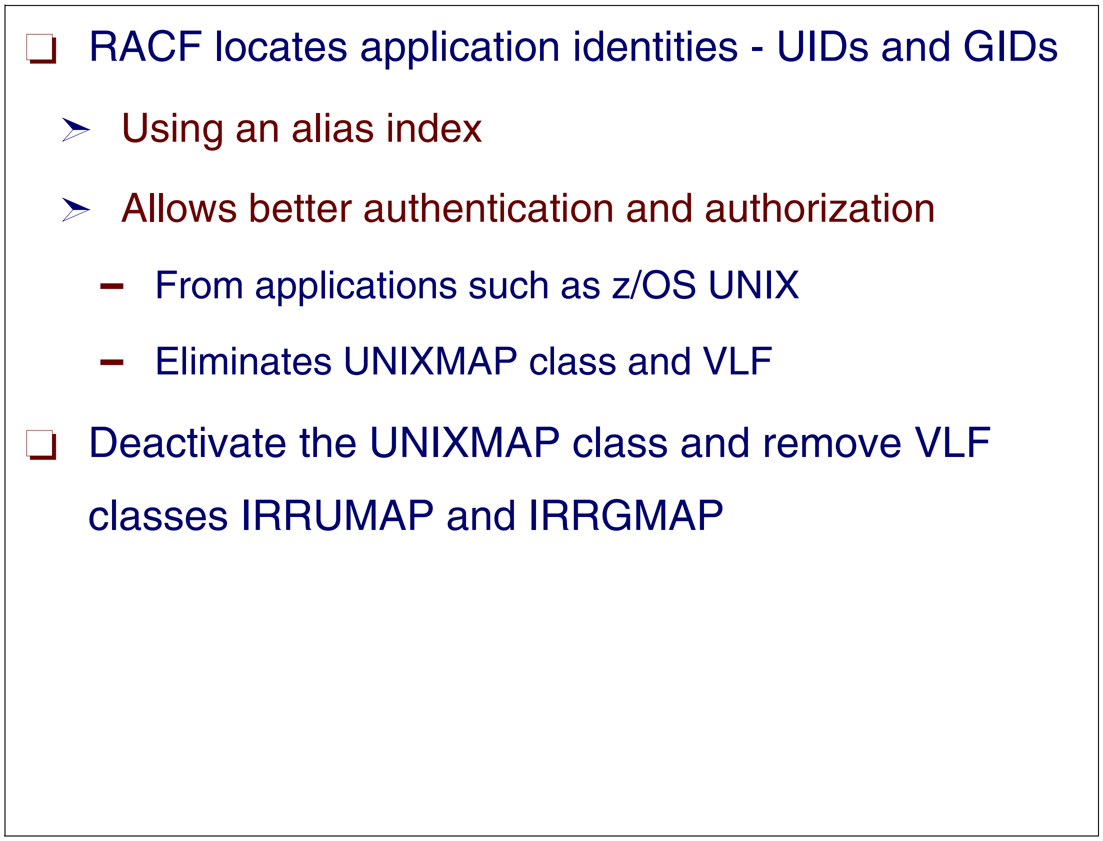
This example shows the definitions to add to the COFVLFxx member in SYS1.PARMLIB to support the RACF UMAP, GMAP, and SMAP tables. You are strongly urged to use IRRUMAP and IRRGMAP for any installation. Samples are also included in the member RACPARM in SYS1.SAMPLIB which is delivered with RACF.

It is also recommended for extra performance improvements that you cache the z/OS UNIX security packets by adding the IRRSMAP entries to COFVLFxx.

When working with the z/OS UNIX shell or ISHELL, some of the commands or functions can display output with either the UID or userid as the file owner. z/OS UNIX only knows about UIDs and GIDs. Whenever a userid or group name is displayed, it has been looked up in the RACF data base or the mapping tables to find the corresponding userid for a UID, or group for a GID. The end user does not have to be aware that such a mapping/conversion is done.

Something which causes a bit of confusion is which userid RACF will show when a file/directory is owned by a UID=0. In a system, there will be multiple superusers (defined with UID=0, or to the BPX.SUPERUSER class), and RACF can only pick one of these user IDs from the mapping table to show as the owner. It seems like without VLF active, RACF will pick the first superuser user ID, in alphabetical order, that has logged on. With VLF active, RACF will pick the first superuser userid that has logged on since VLF was started. This is not a problem; it just causes some confusion for people to see a file owned by one user ID one day, and another user ID another day. This situation happens only for users that share the same UID, which should only be the superusers.

14.18 AIM Stage 3 and z/OS V1R4



- ❑ RACF locates application identities - UIDs and GIDs
  - ➢ Using an alias index
  - ➢ Allows better authentication and authorization
    - – From applications such as z/OS UNIX
    - – Eliminates UNIXMAP class and VLF
- ❑ Deactivate the UNIXMAP class and remove VLF classes IRRUMAP and IRRGMAP

Figure 14-18 Conversion of the RACF database to AIM

You can convert your RACF database to stage 3 of application identity mapping (AIM) using the IRRIRA00 conversion utility. In stage 3, RACF locates application identities, such as UIDs and GIDs, for users and groups by using an alias index that is automatically maintained by RACF. This allows RACF to more efficiently handle authentication and authorization requests from applications such as z/OS UNIX than was possible using other methods, such as the UNIXMAP class and VLF. Once your installation reaches stage 3 of application identity mapping, you will no longer have UNIXMAP class profiles on your system, and you can deactivate the UNIXMAP class and remove VLF classes IRRUMAP and IRRGMAP.

You can improve RACF performance when looking up UIDs and GIDs by using virtual lookaside facility (VLF) and alias index entries. For more information on using them, see *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683.

Using alias index entries allows you to use the RACF **SEARCH** command to determine which users are assigned a specified UID, and which groups are assigned a specified GID, as follows:

```
SEARCH CLASS(USER) UID(0)
SEARCH CLASS(GROUP) GID(100)
```

14.19 Further tuning tips

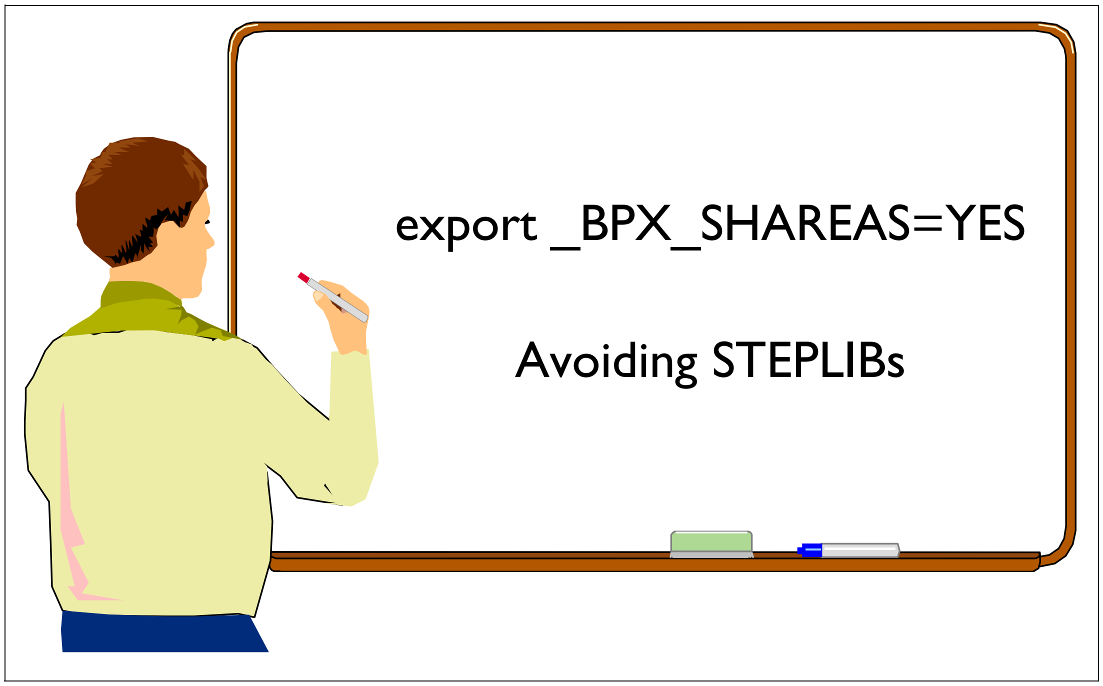


Figure 14-19 Additional tuning tips

To improve z/OS UNIX shell performance, it is recommended that you do the following:

- There are two environment variables that can be used to improve the performance:
  - Set the _BPX_SHAREAS environment variable to YES or REUSE. This saves the overhead of a fork and exec by not creating its own address space for foreground shell processes.
  - Set the _BPX_SPAWN_SCRIPT environment variable to YES. This avoids having the shell invoke spawn after receiving ENOEXEC for an input shell script.
- To improve the performance for all shell users, the /etc/profile should contain the following settings:

```
export _BPX_SHAREAS=YES (or export _BPX_SHAREAS=REUSE)
export _BPX_SPAWN_SCRIPT=YES
```

- Avoid STEPLIBs

  To improve performance for users who log in to the shell with the OMVS command, do not place any STEPLIB or JOBLIB DD statements in logon procedures. Specify STEPLIB=none in the /etc/profile to avoid excessive searching of STEPLIB data sets. The section in the /etc/profile, that contains the STEPLIB statement, should look like:

```
if  -z "$STEPLIB"  &&; tty -s; then
    echo "- Improve performance by preventing the propagation of -"
    echo "- TSO/E or ISPF STEPLIBs                               -"
    export STEPLIB=none
    exec sh -L
fi
```

- Be aware of storage consumption. If the system is running in an LPAR or as a VM guest, the storage size should be at least 64 MB; however, having quite a bit more than this will not be harmful.

ECSA storage used by z/OS UNIX is based on the following formula:

(n * 150 bytes) + (m * 500 bytes)

where n is the number of tasks using z/OS UNIX and m is the number of processes.

So if your system supports 500 processes and 2000 threads, z/OS UNIX consumes 550 KB of ECSA storage.

In addition to this:

– WM uses some ECSA for each forked initiator.
– The OMVS address space itself uses 20KB ECSA.
– Spawn usage requires approximately 100 KB of ECSA.
– Each process that has a STEPLIB that is propagated from parent to child or across an EXEC consumes about 200 bytes of ECSA.

Taking these points into consideration may prevent possible storage shortages.

14.20 SMF recording

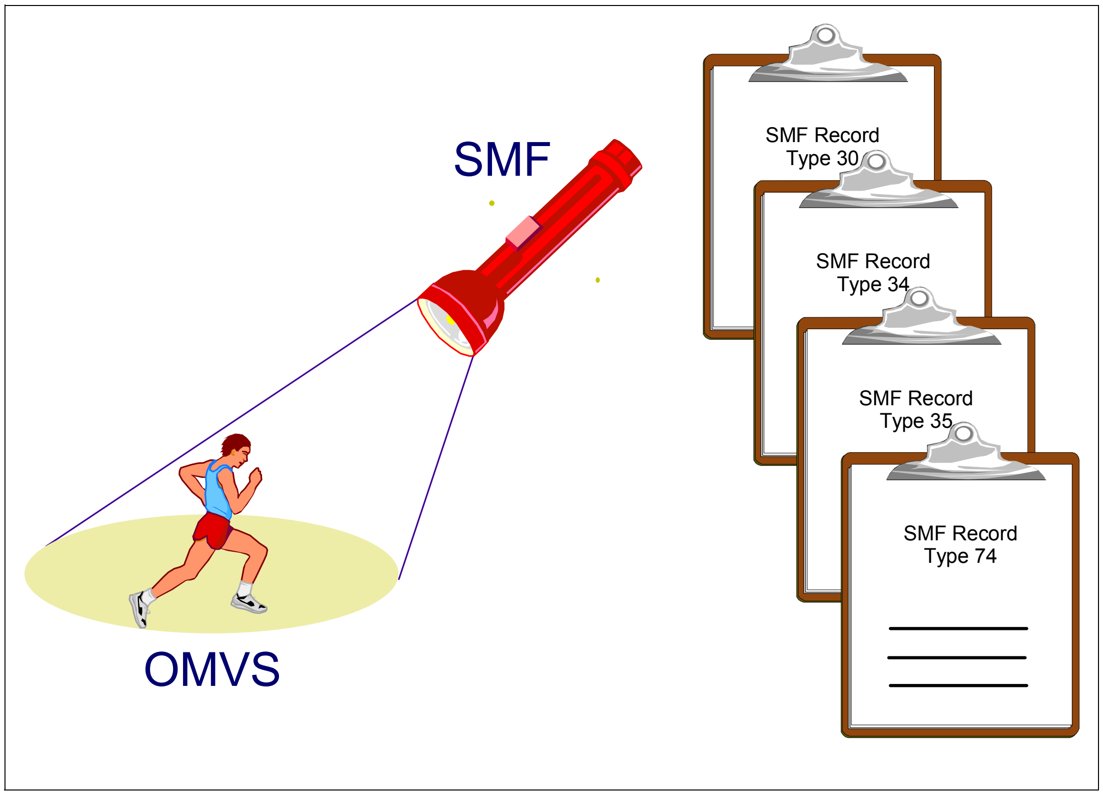


Figure 14-20 SMF recording of z/OS UNIX processing

You can use SMF to report on activity from a user application, to report activity on a job and jobstep basis, and to report the activity of mounted file systems and files.

SMF records are written to store information about the following activities:

- SMF record type 30

  SMF record type 30 reports activity on a job and jobstep basis. 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

- SMF record type 34 and 35

  When a new address space is created for a fork or spawn, SMF cuts a type 34 record. When the process ends, SMF cuts a type 35 record. A type 34 record is defined as TSO/E logon and a type 35 record is defined as TSO/E logoff. If these records are not active in the environment, no further actions are necessary. If they are active for TSO/E accounting, it is necessary (recommended) to suppress these records for UNIX processes. To suppress type 34 and type 35 records, add the following to the SMFPRMxx parmlib member:

  ```
  SYS(TYPE(34,35)) SUBSYS(OMVS,NOTYPE(34,35))
  ```

- SMF record type 74

  SMF record type 74, subtype 3, reports kernel activity.
- SMF record type 80

  SMF record type 80 includes an extended length relocate section.
- SMF record type 92

  SMF record type 92 reports the activity of mounted file systems and files. I/O activity data from an entire mounted file system is provided only when the file system is unmounted. However, these records are useful because they provide information on the total space available in the file system and the total space currently used. This provides an indication of when it is time to increase the size of a mountable file system.

14.21 RMF reporting

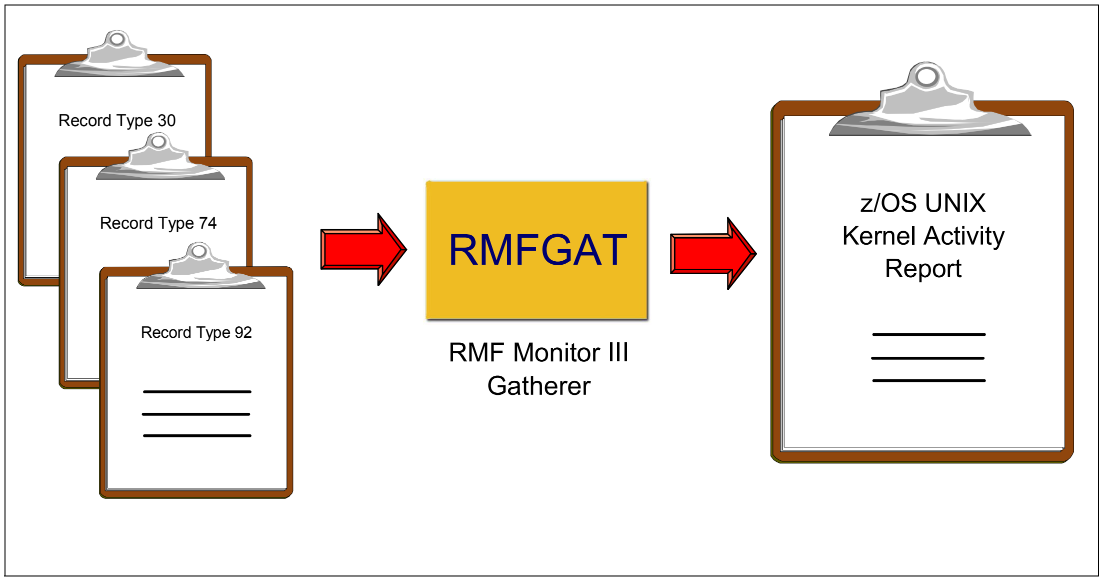


Figure 14-21 RMF reporting of z/OS UNIX processing

The Resource Measurement Facility (RMF) provides z/OS UNIX report information using existing records as described previously.

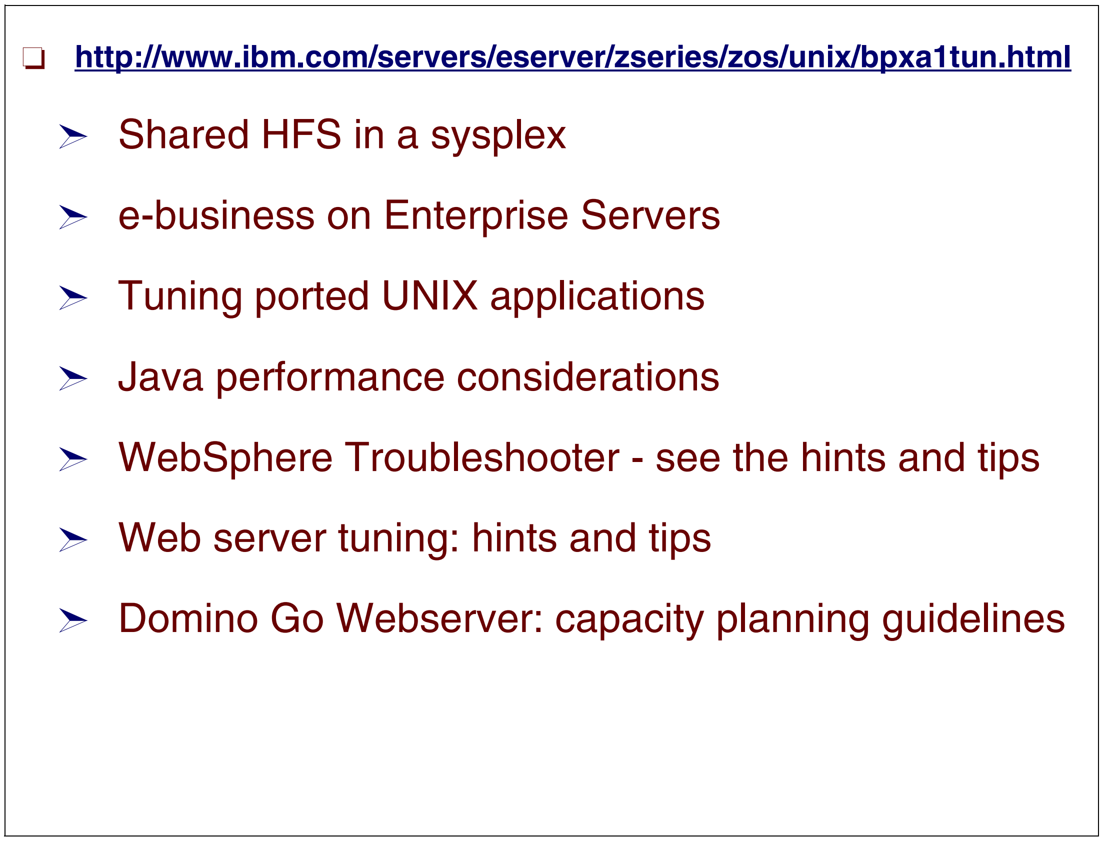
RMF invokes the Monitor III procedure RMFGAT to obtain z/OS UNIX data. The RMFGAT started task must be associated with a user ID that has an OMVS segment. The following RACF command is usable to give RMFGAT a UID and to designate the root directory as its home directory:

```
ADDUSER RMFGAT DFLTGRP(<OMVSGROUP>) OMVS(UID(4711) HOME('/'))
```

Gathering options for z/OS UNIX are not included in the default parmlib member for RMF Monitor I. z/OS UNIX data is gathered by Monitor III, and not by Monitor I.

The Monitor III data gatherer collects z/OS UNIX data for input to the RMF post processor. This data can be used to create a z/OS UNIX kernel activity report.

14.22 z/OS UNIX internet information



- **http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1tun.html**
  - Shared HFS in a sysplex
  - e-business on Enterprise Servers
  - Tuning ported UNIX applications
  - Java performance considerations
  - WebSphere Troubleshooter - see the hints and tips
  - Web server tuning: hints and tips
  - Domino Go Webserver: capacity planning guidelines

Figure 14-22 Where to find information about z/OS UNIX on the internet

Figure 14-22 shows the Web address for information about performance considerations in various areas of z/OS UNIX.

15

Printing services for z/OS UNIX

This chapter discusses how printing requirements are changing, explains why print consolidation with z/OS is the best way to handle printing, and describes how the Infoprint Server supports printing in the z/OS environment for z/OS UNIX users from the shell.

15.1 How do I print and where

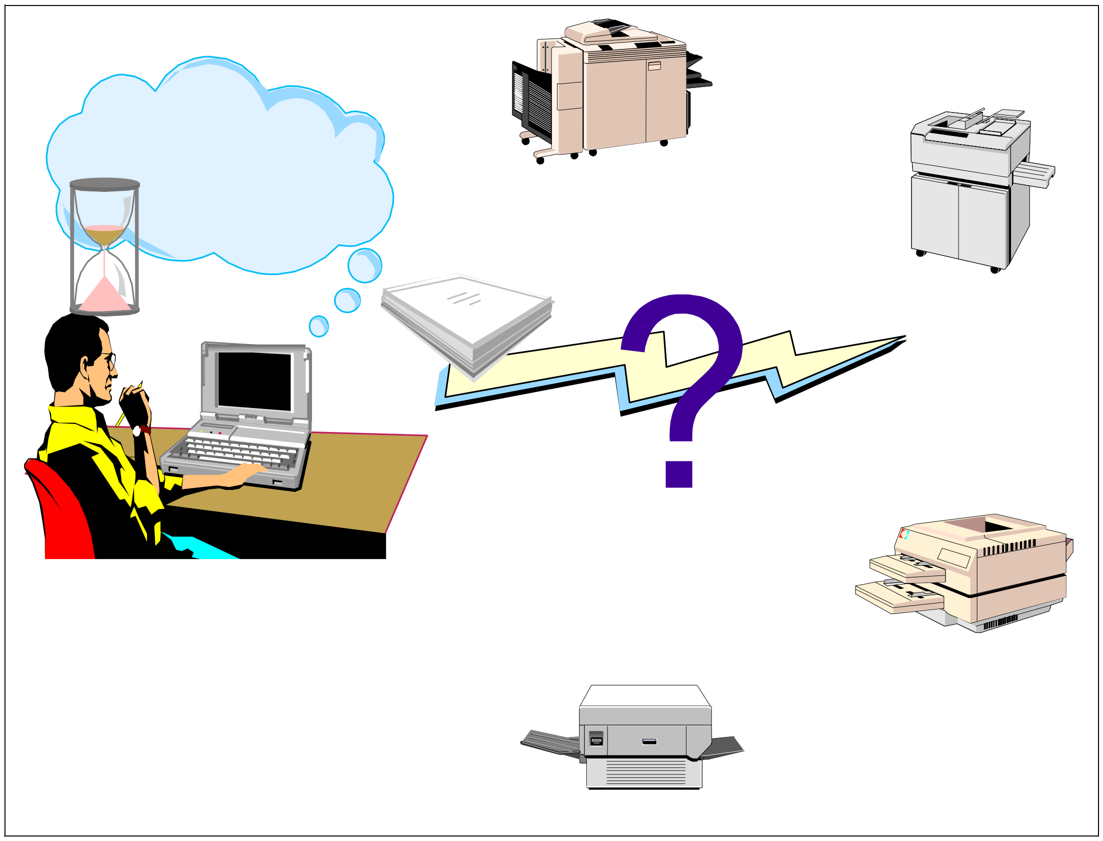


Figure 15-1 Where and how a z/OS UNIX user prints

In a modern environment the possibilities for printing are endless. The question is not only to find the right path to the printer, but also how fast the printer is, the transmission to the printer, and which printer can be used. Normally the user who wants to print a document is not concerned about how his print job reaches the printer.

Infoprint Server is the framework for a total print serving solution for the z/OS system environment. It lets you use the right printer for specific print jobs, balance print workload across all available printers, and more easily manage the inventory of printers.

15.2 z/OS Infoprint Server

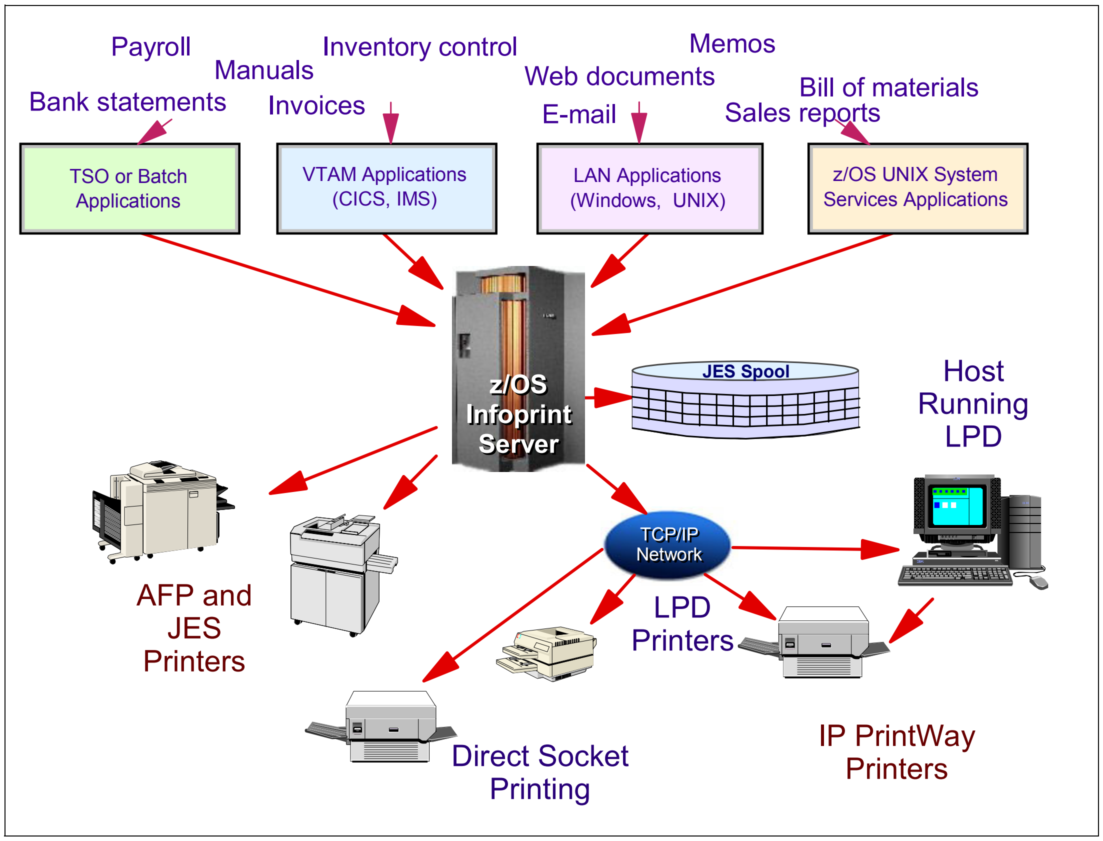


Figure 15-2 Overview of the Infoprint Server

Infoprint Server is an optional feature of OS/390 Version 2 Release 8 and higher, and z/OS Version 1 Release 1 and higher. Infoprint Server is a UNIX application that uses OS/390 UNIX System Services in OS/390 systems and z/OS UNIX System Services in z/OS systems. This feature is the basis for a total print serving solution for the OS/390 or z/OS environment in a TCP/IP network.

Infoprint Server lets users submit print requests from remote workstations in a TCP/IP network, from UNIX System Services applications, from batch applications, and from VTAM applications, such as CICS or IMS applications. It allows you to consolidate your print workload from the servers onto a central z/OS print server as shown in Figure 15-2.

To give the end user more comfort and more options for printing, the z/OS Infoprint Server was introduced with OS/390 V2 R8. This new optional feature gives users the opportunity to consolidate print workloads on z/OS. It allows access to fast and reliable AFP™ printers, JES printers, or TCP/IP connected printers from z/OS, including UNIX services and LAN clients.

Users can define their printers in a central repository allowing clients in the network to use any printer in the enterprise that is registered to the z/OS Infoprint Server.

The z/OS Infoprint Server also provides support for Windows 95 and Windows NT® operating systems and for z/OS UNIX.

15.3 Infoprint Server components

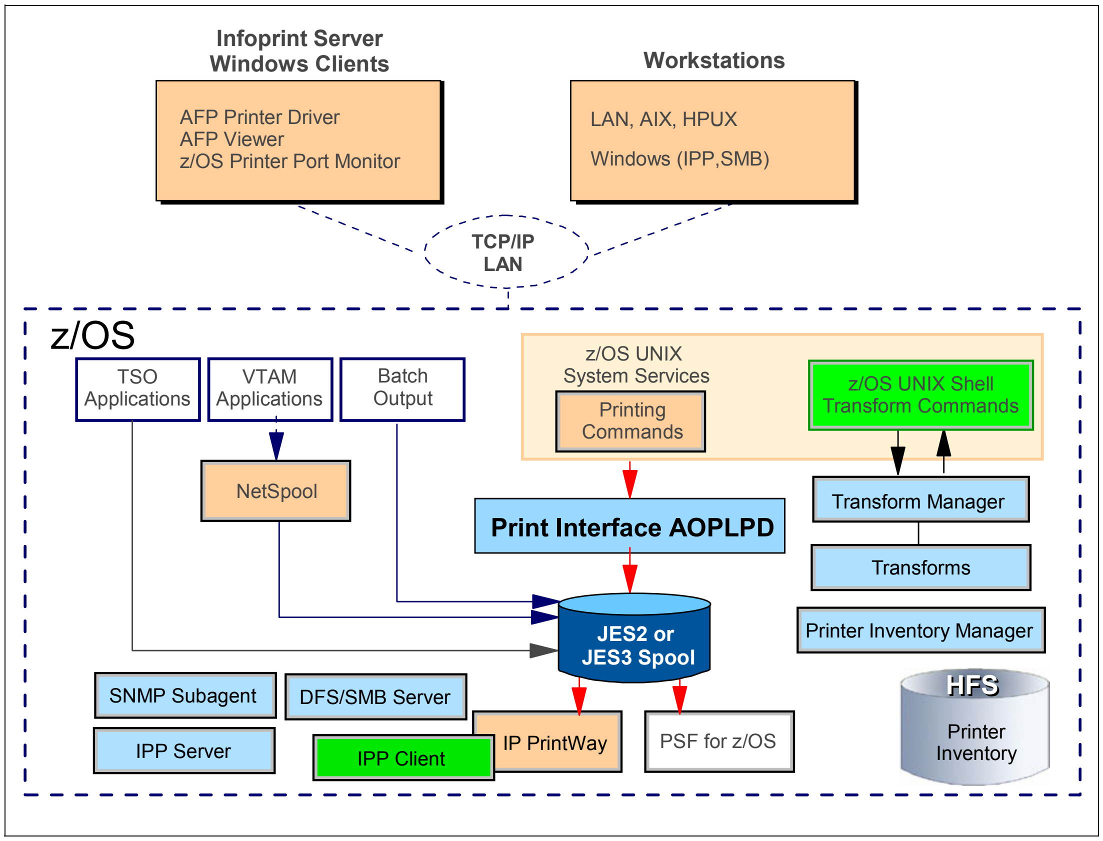


Figure 15-3 Infoprint Server components

Figure 15-3 shows the components of Infoprint Server and how they fit into your operating system. The key components for z/OS UNIX users are:

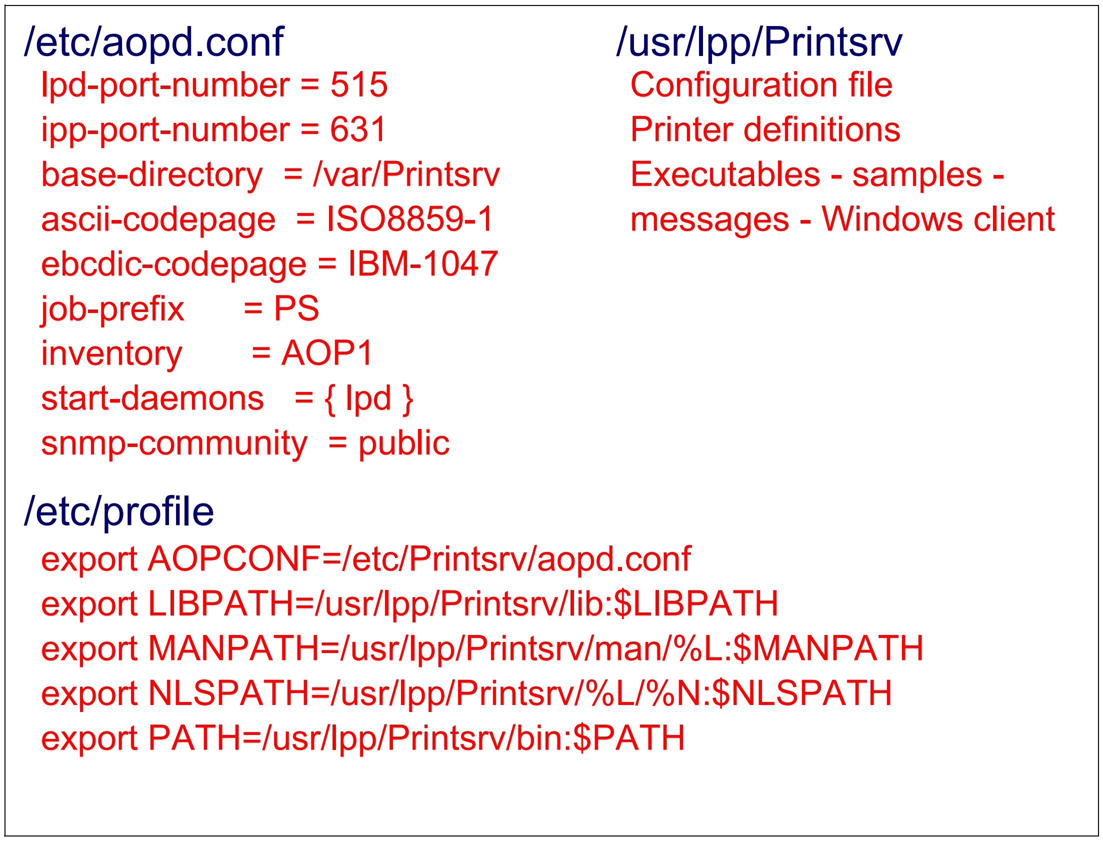
Print Interface Print Interface is the component of the Infoprint Server that accepts input from remote workstations that have TCP/IP access, and from z/OS UNIX System Services printing commands, and creates output data sets on the JES spool.

Printer Inventory The Printer Inventory contains information about both local and remote printers and is maintained by the system administrator using an ISPF application. When a user sends data to be printed, the printer definition in the Printer Inventory is used to determine where to print the data.

Print commands A z/OS UNIX Services user can print files using the Print Interface Services. Print Interface provides enhanced versions of the z/OS UNIX System Services shell printing commands. These commands have more functions than the standard UNIX shell printing commands.

Transforms An administrator can set up the transforms to automatically convert data when printing. A user can also transform documents to and from the AFP data format from the z/OS UNIX command line. Documents transformed from the command line can be saved in the converted format and printed later or sent to other users.

15.4 Installation of Infoprint Server



/etc/aopd.conf

```
lpd-port-number = 515
ipp-port-number = 631
base-directory  = /var/Printsrv
ascii-codepage  = ISO8859-1
ebcdic-codepage = IBM-1047
job-prefix      = PS
inventory       = AOP1
start-daemons   = { lpd }
snmp-community  = public
```

/usr/lpp/Printsrv

Configuration file
Printer definitions
Executables - samples - messages - Windows client

/etc/profile

```
export AOPCONF=/etc/Printsrv/aopd.conf
export LIBPATH=/usr/lpp/Printsrv/lib:$LIBPATH
export MANPATH=/usr/lpp/Printsrv/man/%L:$MANPATH
export NLSPATH=/usr/lpp/Printsrv/%L/%N:$NLSPATH
export PATH=/usr/lpp/Printsrv/bin:$PATH
```

Figure 15-4 Customizing the Infoprint Server

The Infoprint Server configuration file, aopd.conf, lets you customize the Printer Inventory Manager and other components of Infoprint Server. This file is optional. If the configuration file does not exist or if an attribute in the configuration file is omitted, default values are used.

Infoprint Server environment variables can be set in these two locations:

- In the aopstart EXEC: The Printer Inventory Manager, and other Infoprint Server daemons, use environment variables specified in this file.
- In the /etc/profile file: The z/OS UNIX printing commands and other commands, such as **`lp`**, **`pidu`**, and **`ps2afp`**, use environment variables specified in this file.

To edit the /etc/profile file, you can use the TSO/E **`OEDIT`** command or the z/OS UNIX **`oedit`** command. To set and export an environment variable, use the z/OS UNIX **`export`** command. For example, if you installed Infoprint Server libraries in the default locations, add these commands to the /etc/profile file.

As shown in Figure 15-5 on page 560, the following customization steps should be done to configure the HFS and the configuration files you require for your installation:

1. Create the /etc/Printsrv directory. You can use the UNIX **`mkdir`** command under the /etc directory or the ISHELL to create the /etc/Printsrv directory. This directory is the default location for the Infoprint Server configuration files, aopd.conf, aopxfd.conf, and aopsapd.conf.

2. Create the /var/Printsrv directory. Issue the UNIX **mkdir** command or use the ISHELL to create the directory. If you do not create this directory, the **aopsetup** shell script will create it.
3. Set up the Infoprint Server configuration files for use. Copy the sample configuration files from /usr/lpp/Printsrv/samples to the default location /etc/Printsrv/ with the z/OS UNIX cp command or use the ISPF ISHELL. You can choose to copy the configuration file into another location; however, if you do, specify the full path name of the configuration file in the AOPCONF environment variable in the /etc/profile file.

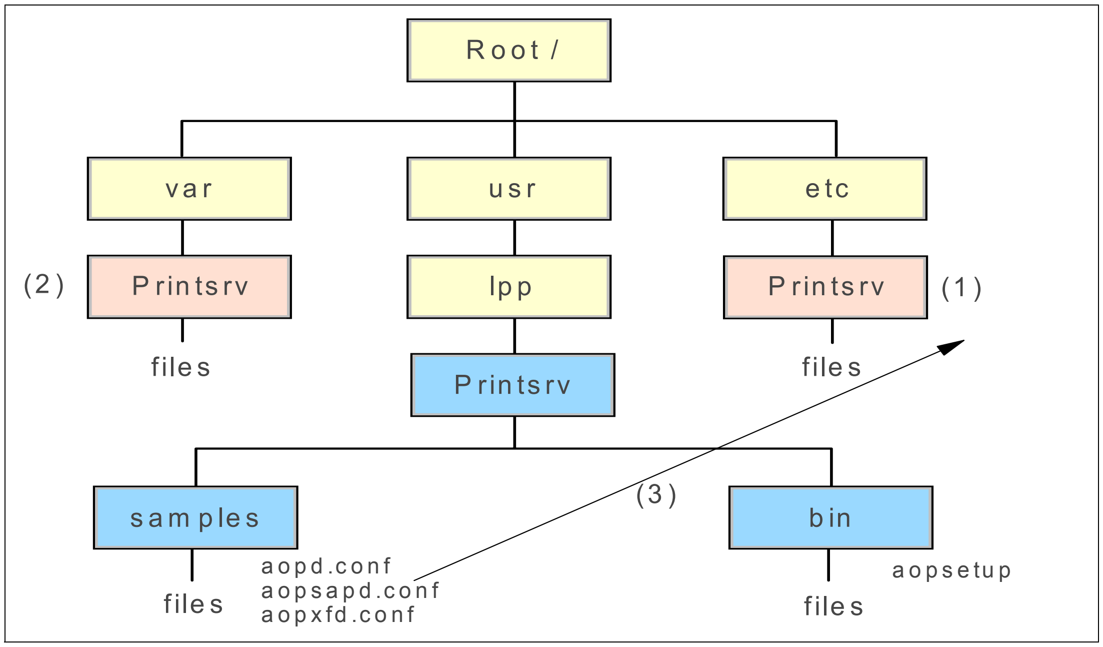
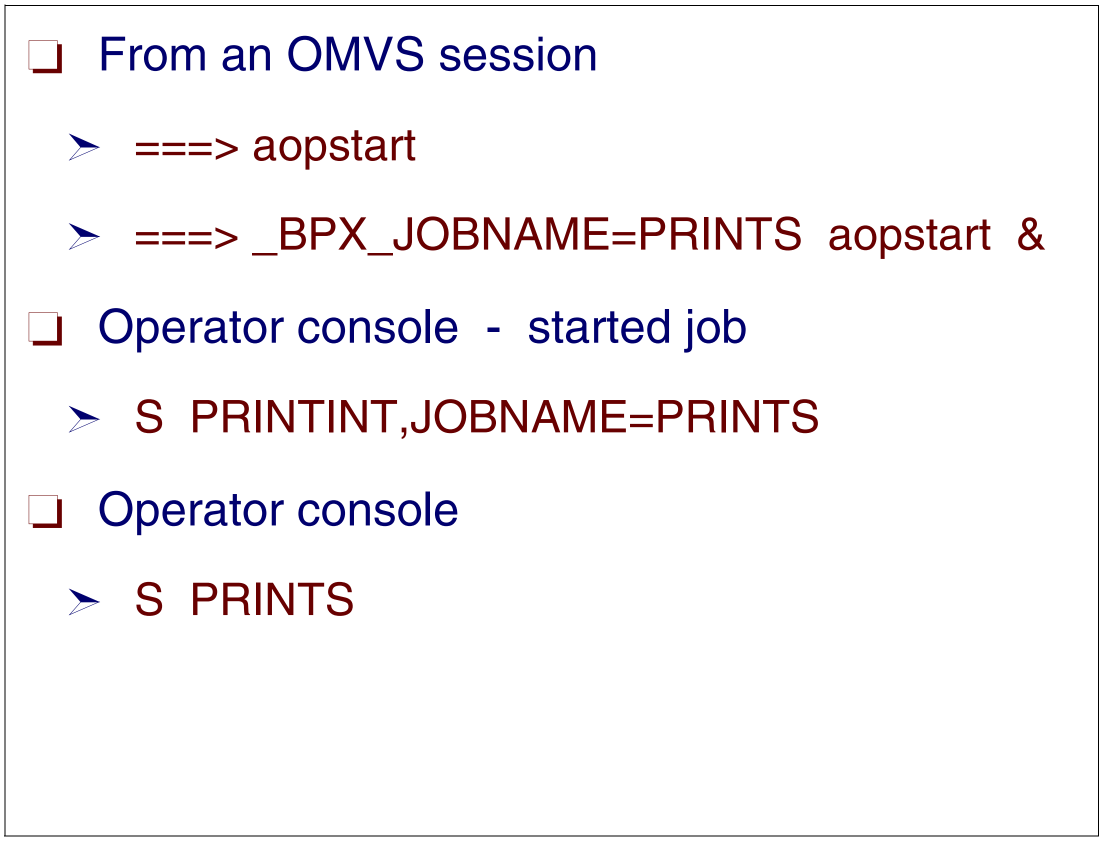


Figure 15-5 HFS directory for Infoprint Server directories and files

15.5 Starting Print Interface



❑ From an OMVS session

➢ ===> aopstart

➢ ===> _BPX_JOBNAME=PRINTS aopstart &

❑ Operator console - started job

➢ S PRINTINT,JOBNAME=PRINTS

❑ Operator console

➢ S PRINTS

Figure 15-6 Starting the Infoprint Server (Print Interface)

We would recommend that you start the Print Interface by using a started task or by using the BPXBATCH utility. There are three ways to start the Print Interface:

- From OMVS, issue the **`aopstart`** command
- Operator issues start task command
- Operator issues a started job command

The **`aopstart`** command starts the Printer Inventory Manager daemon, aopd. It also starts any other Infoprint Server daemons specified in the start-daemons attribute in the aopd.conf configuration file.

The **`aopstop`** command stops either the Printer Inventory daemon and any other active Infoprint Server daemons, or it stops only selected Infoprint Server daemons, depending on the command options.

If you have set up all the Printer Inventory daemon and Infoprint Server daemon environment variables in /etc/profile, the **`aopstart`** and **`aopstop`** commands can be entered as follows:

- From the z/OS UNIX shell, where you can enter **`aopstart`** and **`aopstop.`**
- As a recommended alternative, beginning with OS/390 Release 8, you can use JCL procedures to invoke the **`aopstart`** and **`aopstop`** commands. The JCL procedures in SYS1.IBM.PROCLIB that Infoprint Server ships are named AOPSTART and AOPSTOP.

Add the **`aopstart`** command to the /etc/rc shell script to start the Printer Inventory Manager (and other daemons) automatically during the IPL.

15.6 Operator console started job

```
//PRINTINT JOB ' ','ROGERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
//         REGION=4M,TIME=1440,NOTIFY=&SYSUID
//*****************************************************************
//* RUN THE z/OS Print Interface FROM BATCH as a STARTED JOB
//*    S PRINTINT,JOBNAME=PRINTS
//*****************************************************************
//STEP1   EXEC PGM=BPXBATCH,  <-----------------
//   PARM='PGM /usr/lpp/Printsrv/bin/aopstart'
//STDOUT   DD PATH='/tmp/Printsrv-stdout',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=SIRWXU
//STDERR   DD PATH='/tmp/Printsrv-stderr',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=SIRWXU
//STDENV   DD *
AOPCONF=/etc/Printsrv/aopd.conf
PATH=/usr/lpp/Printsrv/bin
LIBPATH=/usr/lpp/Printsrv/lib
MANPATH=/usr/lpp/Printsrv/man/C
NLSPATH=/usr/lpp/Printsrv/en_US/%N
TZ=EST5EDT
/*
```

Figure 15-7 Using a batch job to start the Infoprint Server

You can start the Print Interface using a JCL started job with BPXBATCH.

IBM recommends that you use AOPBATCH instead of BPXBATCH to run programs provided by Infoprint Server because AOPBATCH sets default values for the PATH, LIBPATH, and NLSPATH environment variables that are suitable for installations that installed Infoprint Server files in default locations. Also, AOPBATCH lets STDIN be read from a DD statement and lets STDOUT and STDERR be written to a DD statement.

AOPBATCH lets you use MVS job control language (JCL) to run a program that resides in a hierarchical file system (HFS).

15.7 Operator console started task

```
//PRINTS   PROC
//PRINTS   EXEC PGM=BPXBATCH,
//   PARM='PGM /usr/lpp/Printsrv/bin/aopstart'
//STDOUT   DD PATH='/tmp/Printsrv-stdout',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=SIRWXU
//STDERR   DD PATH='/tmp/Printsrv-stderr',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=SIRWXU
//STDENV   DD PATH='/etc/Printsrv.envvars',
//          PATHOPTS=ORDONLY
```

/etc/Printsrv.envvars

```
AOPCONF=/etc/Printsrv/aopd.conf
PATH=/usr/lpp/Printsrv/bin
LIBPATH=/usr/lpp/Printsrv/lib
MANPATH=/usr/lpp/Printsrv/man/C
NLSPATH=/usr/lpp/Printsrv/en_US/%N
TZ=EST5EDT
_BPXK_SETIBMOPT_TRANSPORT=TCPIPOE
```

Figure 15-8 Starting Infoprint Server from SYS1.PROCLIB

You may want to execute your Print Interface application using JCL in SYS1.PROCLIB that executes the BPXBATCH utility. BPXBATCH is an MVS utility that you can use to run shell commands or shell scripts and to run executable files through the MVS batch environment.

With BPXBATCH, you can allocate the MVS standard files stdin, stdout, and stderr as HFS files. If you do allocate these files, they must be HFS files. You can also allocate MVS data sets or HFS text files containing environment variables (stdenv). If you do not allocate them, stdin, stdout, stderr, and stdenv default to /dev/null. Allocate the standard files using the data definition PATH keyword options, or standard data definition options for MVS data sets, for stdenv.

The environment variables that are needed to run the Print Interface using BPXBATCH were defined in a file in the HFS.

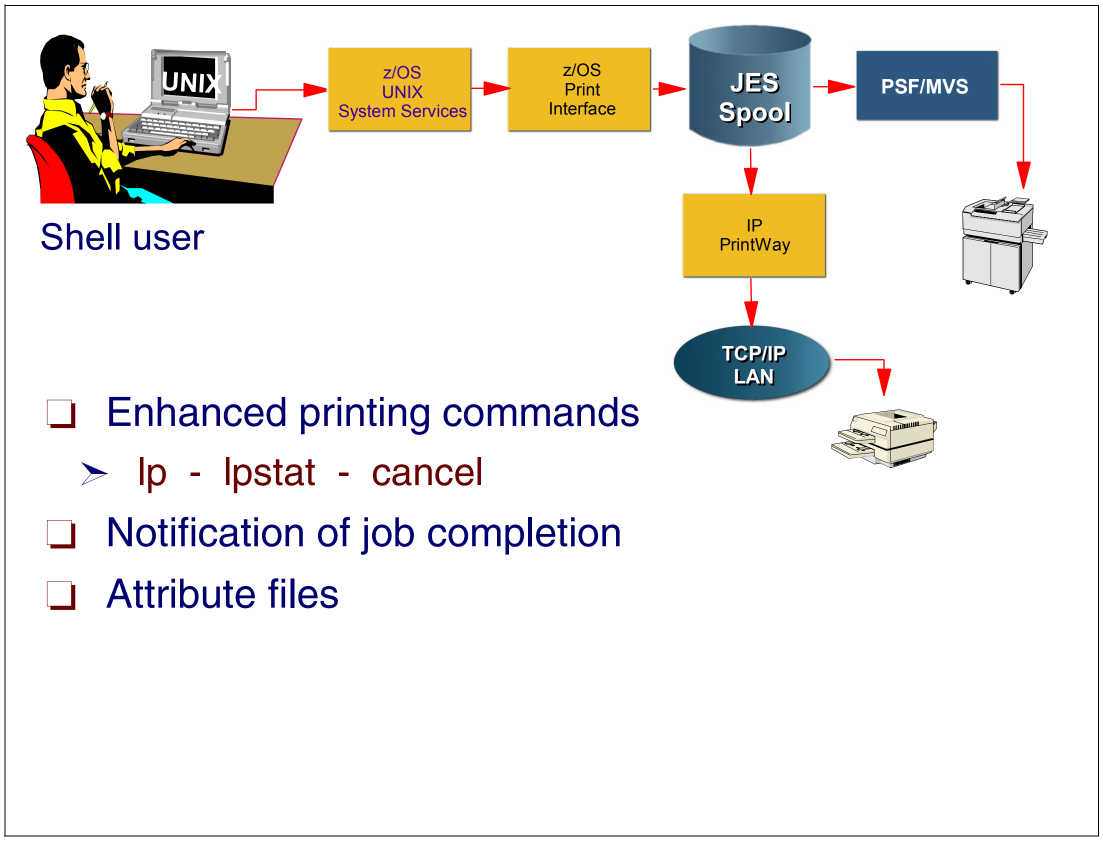
To start the Print Interface from an operator command, you can create a procedure in your PROC library. You place the procedure, shown in Figure 15-8, in member PRINTS in the SYS1.PROCLIB data set.

As you see in the STDENV DD statement and in the visual, we placed the environment variables in file /etc/Printsrv.envvars. To start the Print Interface, issue the following operator command:

```
S PRINTS
```

When the task executes, it creates a forked address space.

15.8 Printing from UNIX System Services



- ❑ Enhanced printing commands
  - ➢ lp - lpstat - cancel
- ❑ Notification of job completion
- ❑ Attribute files

Figure 15-9 Printing from the z/OS UNIX shell

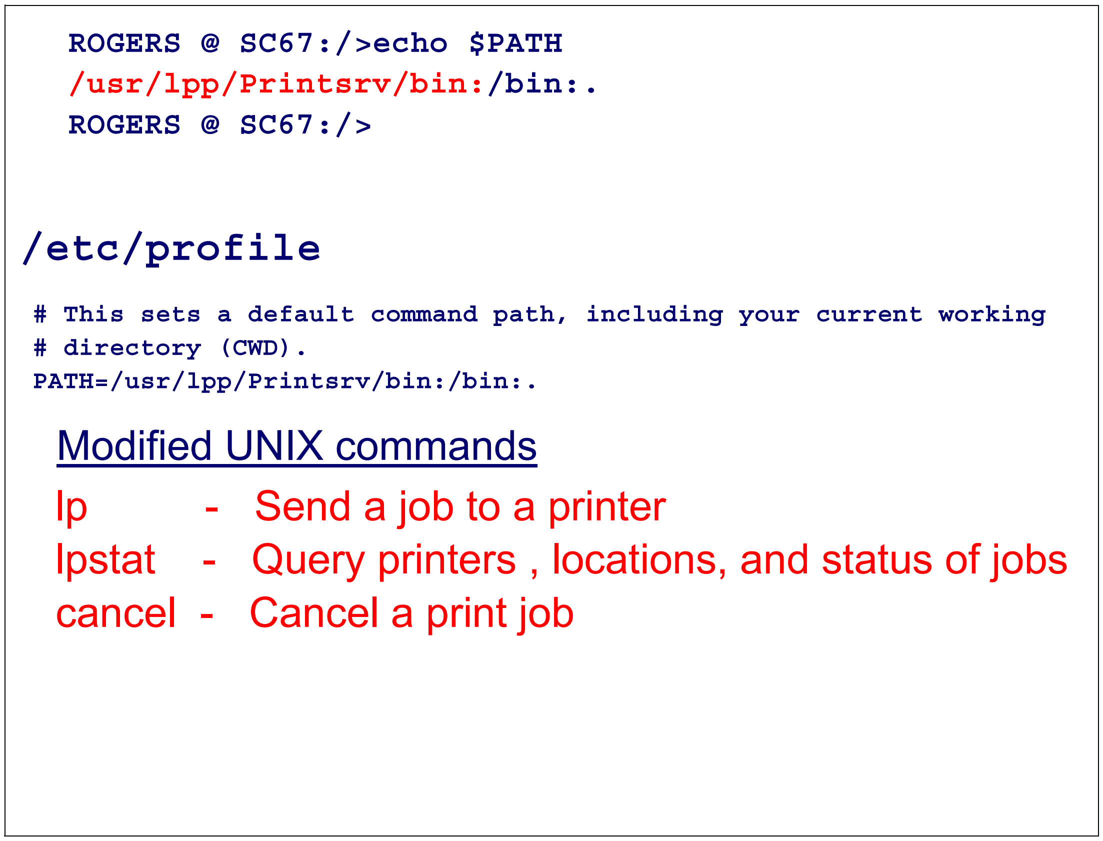
From the z/OS UNIX shell you can print to any printer defined in the Printer Inventory of the z/OS Infoprint Server. You can print on local printers attached directly to z/OS, or on remote printers in a TCP/IP LAN network.

Your system administrator assigns a name to each printer defined in the Printer Inventory. To print, you need to know this name. A printer in the Printer Inventory can be a physical printer or a pool of physical printers that can print the same types of files. Or your administrator can define more than one printer name for the same physical printer, so that you can use a different printer name for printing files with different characteristics.

The z/OS Infoprint Server attempts to validate that your file can print on the selected printer before accepting your print request. For example, if the printer you select cannot print the type of data (PostScript, PCL, and so on) in your file, the z/OS Infoprint Server does not accept your request and sends you a message.

The z/OS UNIX printing commands provided by Infoprint Server, in /usr/lpp/Printsrv/bin, have enhanced function over the commands of the same name described in the *z/OS UNIX System Services Command Reference*, SA22-7802. For example, when printing on AFP printers, you can specify options such as duplexing or a special overlay. You can also query the status of your print request, and you can cancel a print request. These printing commands adhere to the UNIX standards in XPG4.2, so that you do not need to change your UNIX applications when you port them to z/OS.

15.9 UNIX commands with Infoprint Server



```
ROGERS @ SC67:/>echo $PATH
/usr/lpp/Printsrv/bin:/bin:.
ROGERS @ SC67:/>
```

/etc/profile

```
# This sets a default command path, including your current working
# directory (CWD).
PATH=/usr/lpp/Printsrv/bin:/bin:.
```

Modified UNIX commands

lp	-	Send a job to a printer
lpstat	-	Query printers , locations, and status of jobs
cancel	-	Cancel a print job

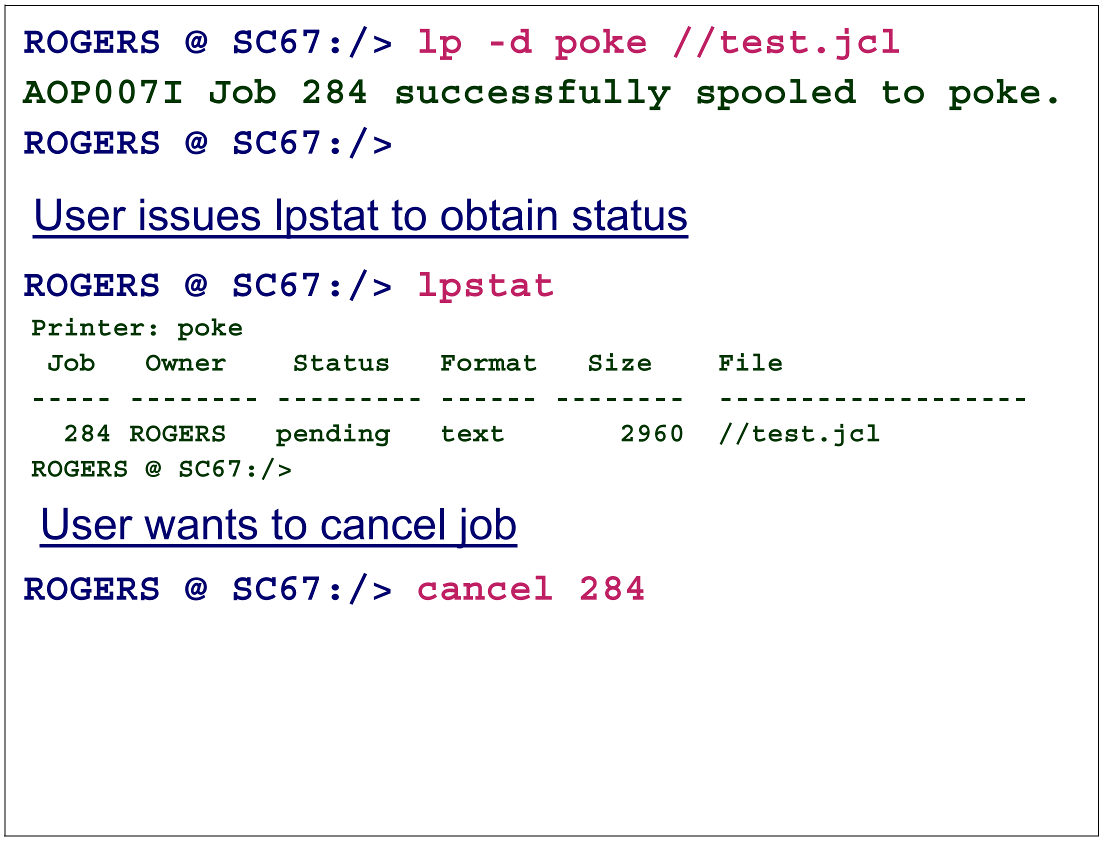
Figure 15-10 z/OS UNIX shell commands modified by Infoprint Server

The `lp`, `lpstat`, and `cancel` commands, shown in Figure 15-10, use TCP/IP protocol to send print requests to the Print Interface. They send commands to the port number specified in the Print Interface configuration file.

The `lp`, `lpstat`, and `cancel` commands are modified to be used with the Print Interface and are placed in the HFS as follows:

/usr/lpp/Printsrv/bin

15.10 UNIX user prints a data set



```
ROGERS @ SC67:/> lp -d poke //test.jcl
AOP007I Job 284 successfully spooled to poke.
ROGERS @ SC67:/>
```

User issues lpstat to obtain status

```
ROGERS @ SC67:/> lpstat
 Printer: poke
  Job   Owner     Status    Format   Size     File
 ----- -------- --------- ------ --------  ------------------
   284 ROGERS   pending   text       2960  //test.jcl
 ROGERS @ SC67:/>
```

User wants to cancel job

```
ROGERS @ SC67:/> cancel 284
```

Figure 15-11 An example of a user printing a data set from the shell

UNIX System Services users can use the **lp** command to print data sets to printers defined to the Print Interface. The UNIX user issues the **lp** command, as shown in Figure 15-11, specifying **poke** as the Print Interface defined printer shown in visual. The data set to be printed is an MVS data set: TEST.JCL

The z/OS UNIX shell commands are used as follows:

- Sending a job to print or to send one or more files to print, use the **lp** command. For example, to print three copies of myfile1 and myfile2 on Printer2, enter:

  ```
  lp -d Printer2 -n 3 myfile1 myfile2
  ```

- To find out where the printers are:

  Use the **lpstat** command to query printer names and locations. For example, to see the names and locations of all printers known to the Printer Inventory, enter:

  ```
  lpstat -a
  ```

- To find out if a job is printing:

  You can use the **lpstat** command to query the status of a job. For example, you submitted several jobs to print and want to know if any of them are printing. To query all your jobs submitted to any printer, enter:

  ```
  lpstat
  ```

- To cancel a job:

Use the **cancel** command to cancel a job. For example, you realize that you need to make some changes in the file that you just sent to print on Printer3. If you don't remember the job ID that the **lp** command returned, use the **lpstat** command to query all the jobs that you submitted to Printer3:

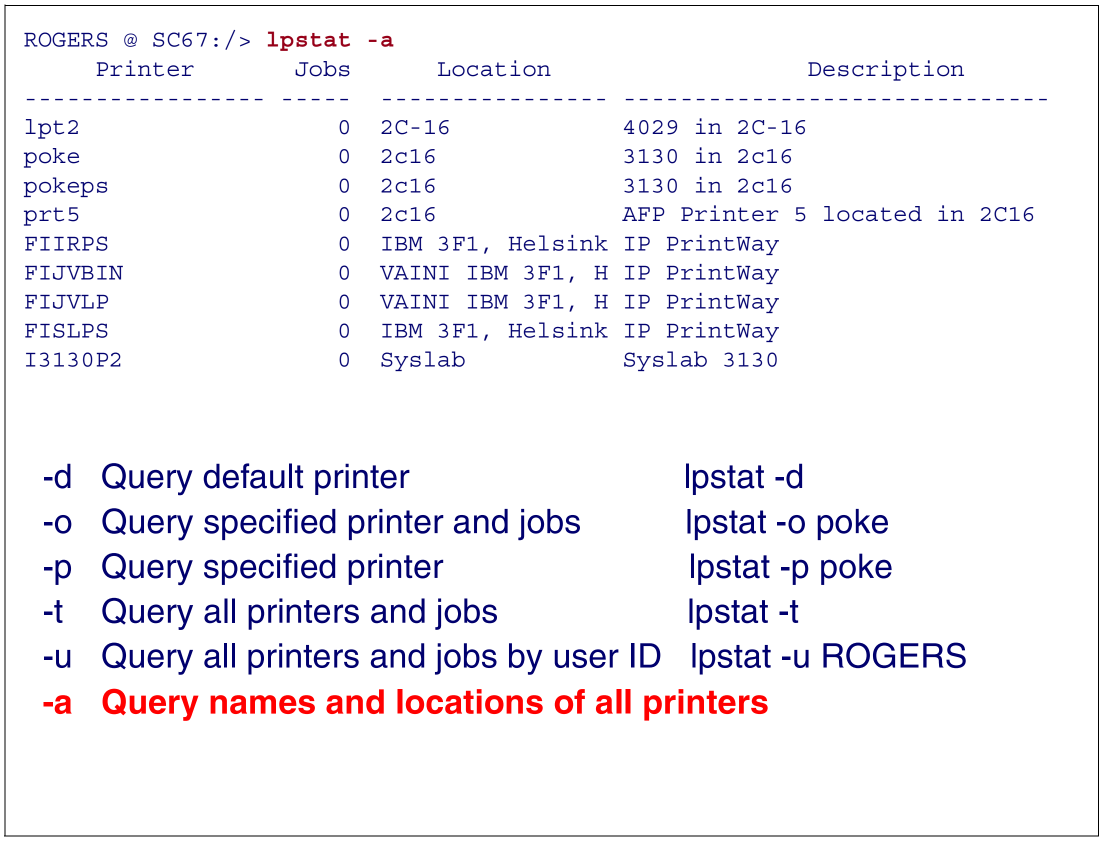
Make sure that the correct path to the commands is in /etc/profile as follows:

```
PATH=/usr/lpp/Printsrv/bin:/bin:
lpstat -o Printer3
```

Suppose that your job has an ID of 284. To cancel it, enter:

```
cancel 284
```

15.11 UNIX user issues the lpstat command



```
ROGERS @ SC67:/> lpstat -a
     Printer       Jobs      Location                  Description
----------------- -----  ---------------- ------------------------------
lpt2                  0  2C-16            4029 in 2C-16
poke                  0  2c16             3130 in 2c16
pokeps                0  2c16             3130 in 2c16
prt5                  0  2c16             AFP Printer 5 located in 2C16
FIIRPS                0  IBM 3F1, Helsink IP PrintWay
FIJVBIN               0  VAINI IBM 3F1, H IP PrintWay
FIJVLP                0  VAINI IBM 3F1, H IP PrintWay
FISLPS                0  IBM 3F1, Helsink IP PrintWay
I3130P2               0  Syslab           Syslab 3130
```

-d	Query default printer	lpstat -d
-o	Query specified printer and jobs	lpstat -o poke
-p	Query specified printer	lpstat -p poke
-t	Query all printers and jobs	lpstat -t
-u	Query all printers and jobs by user ID	lpstat -u ROGERS
-a	**Query names and locations of all printers**	

Figure 15-12 A user issues a command to display submitted output data sets

When a UNIX user needs to know which printers are defined for use, the **lpstat** command can be used as shown in Figure 15-12.

The user can inquire using specific printers, the default printer, or all printers, and request whether or not the jobs waiting for the printers are to be displayed.

15.12 lpstat -t command

```
Printer: poke
 Job   Owner    Status    Format   Size    File
----- -------- --------- ------ --------  ------------------------------
  285 ROGERS   pending   text       3149  ROGERS.TEST.JCL
  286 ROGERS   pending   text       3109  ROGERS.TEST.JCL
  287 TCPIPOE  pending   text       2960  //test.jcl
  288 pc-user  pending   text       2997  test.jcl
  289 ROGERS   pending   text       3108  TEST.JCL
  290 ROGERS2  pending   pcl       20902  ... About the IBM AFP Printer"
  291 ROGERS2  pending   pcl       19019  Printing "Options Dialog"
  296 ROGERS   pending   text       3109  ROGERS.TEST.JCL
  297 ROGERS2  pending   pcl       41436  readme95 - Notepad
  298 ROGERS2  pending   pcl       41436  readme95 - Notepad
  311 ALCIDES  pending   text       2960  //test.jcl

Printer: pokeps
 Job   Owner    Status    Format   Size    File
----- -------- --------- ------ --------  ------------------------------
  292 ROGERS2  pending   modca     19422  scop.AOI
  293 ROGERS2  pending   pcl       41436  readme95 - Notepad
  294 ROGERS2  pending   pcl       41436  readme95 - Notepad
  295 ROGERS2  pending   pcl       41436  readme95 - Notepad
  299 ROGERS2  pending   modca      3650  word.AOI
  301 ROGERS2  pending   modca     19506  word.AOI
  302 ROGERS2  pending   modca     19506  word.AOI
  304 ROGERS2  pending   modca     19422  word.AOI
```

Figure 15-13 Command to display all printers and submitted jobs

To determine which jobs have been submitted to each printer, specify:

```
lpstat -t
```

This command shows all the printers defined to the Print Interface and all the jobs queued to the Print Interface printers. Figure 15-13 only shows two of the printers defined in the Inventory because the amount of information for all printers and jobs was very large.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 572. Note that some of the documents referenced here may be available in softcopy only.

- *Putting the Latest z/OS Security Features to Work*, SG24-6540
- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration*, SG24-5227
- *OS/390 Security Server 1999 Updates: Installation Guide*, SG24-5629

Other publications

These publications are also relevant as further information sources:

- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OZ UNIX System Services User's Guide* , SA22- 7801
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS Using REXX and z/OS UNIX System Services*, SA22-7806
- *z/OS UNIX System Services Messages and Codes*, SA22-7807
- *z/OS UNIX System Services File System Interface Reference*, SA22-7808
- *z/OS MVS System Codes,* SA22-7626
- *z/OS Security Server RACF System Programmer's Guide*, SA22-7681
- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
- *TSM Using the Backup-Archive Clients*, SH26-4105
- *TSM Installing the Clients*, SH26-4102

Online resources

These Web sites and URLs are also relevant as further information sources:

- z/OS UNIX Tools and Toys

  `http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1tun.html`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

ABCs of z/OS System Programming Volume 9

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

ABCs of z/OS System Programming Volume 9

z/OS UNIX, TCP/IP installation, customization

HFS and zFS, Parmlib defintions

Shell and programming tools

The ABCs of z/OS System Programming is a ten volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

This IBM Redbook describes UNIX System Services (z/OS UNIX). It will help you install, tailor, configure, and use the z/OS Version 1 Release 4 version of z/OS UNIX.

SG24-6989-00 ISBN 0738498696